This document was produced by SDC in performance of SD 97

2/22

NOTE *an internal working paper*

System Development Corporation/2500 Colorado Ave./Santa Monica, California

N - 19846/000/00

AUTHOR A. M. Rosenberg

DATE 2/21/63 PAGE 1 OF 3 PAG

ARNOLD I. DUMEY
29 BARBERRY LANE
ROSLYN HEIGHTS, N. Y.

DEVELOPMENTAL AND RESEARCHABLE AREAS IN TIME-SHARING

There exist many unknowns in the practical application of time-sharing system concepts which are not readily approachable until sufficient experience is gained and a time-sharing vehicle is available for study purposes. Research into time-sharing system problems may be initiated when the above criteria are reasonably satisfied and the pressure of the production effort is relieved. However, it is not too early to delineate appropriate considerations for on-going research and development. This note contains an initial set of researchable areas which will be modified as time and experience progress.

1. On-Line Coding - The nature, benefits, and requirements of coding a program statement or a line at a time. Also, the building of programs with library routines.

2. Editing - Techniques for rapid, on-line perusal and manipulation or modification of object program code and/or data. This includes the problem of using source language vs. machine language.

3. Testing - Encompasses all phases of on-line program debugging and the need for appropriate on-line test tools.
   a. Grammar checking
   b. Parameter testing
   c. String testing
   d. System (object) testing
   e. Associating test data with programs

4. Group Interaction - Enabling several users to interact with each other via communication with a common (object) program (system). This implies multiple input control of program response and selective multiple output.

5. Object Program Systems - A methodology whereby object programs may be associated into a single object system with intercommunication between modular entities. One might consider here, the type of programs which lend themselves best to time-sharing.

ARNOLD I. DUMEY
29 BARBERRY LANE
ROSLYN HEIGHTS, N. Y.

6. Dynamic Relocatability - The dynamic assignment of main and auxiliary storage to programs and data. This area will also consider scheduling needs for the hierarchical treatment of available storage.

7. Scheduling and Priority - Optimizing the effectiveness of the Time-Sharing system in servicing various kinds and numbers of user programs and needs, e.g., large programs, small programs, production programs, high data-rate programs, etc. Priority considerations involve time dependent and input dependent response needs.

8. Language Compatibility - The coexistence of different language types within the Time-Sharing system, i.e., POL's, MOL's and list processors. In addition, the optimal language system for time-sharing activities is also researchable.

9. Input (Interrupt) and Output Processing - Involves the problem of overhead efficiency related to rapid response to user needs.

10. Equipment Support - Covers a broad spectrum of areas where hardware can facilitate time-sharing system operations.
    a. Memory protection
    b. Expanding memory storage
    c. Interrupts (time, error, inputs, outputs, etc.)
    d. Display devices
    e. Input keyboard devices
    f. Light pens
    g. Graphical input and output devices
    h. Bulk input and output devices (Remote)

11. Multiple-Computer Operations - The application of time-sharing concepts to multiple-computer and satellite computer configurations.

12. Automated Documentation - Permitting the Time-Sharing system and resident programs to be self-explanatory to any potential user.

13. Applications - Various areas of time-sharing applicability, e.g., data retrieval, learning machines, desk calculator, command planning, language construction, etc.

14. On-Line Technical Management of Time-Sharing System - The problem of proper user utilization of a time-sharing system, particularly for production programming, and user activity accounting and control.

Other researchable areas will appear on the scene as they are conceived or tripped over in the ensuing months of involvement with time-sharing concepts and implementation. There will be an assortment of problem areas which can be investigated as the opportunity arises.

TM- 1050/000/00
AUTHOR A. M. Rosenberg
TECHNICAL
RELEASE J. I. Schwartz
for H. D. Benington
DATE 2/21/63 PAGE 1 OF 4 PAGES

TECH MEMO

a working paper

System Development Corporation / 2500 Colorado Ave. / Santa Monica, California

ARNOLD I. DUMEY
29 BARBERRY LANE
ROSLYN HEIGHTS, N. Y.

TESTING THE ARPA-SDC TIME-SHARING SYSTEM

The design of the initial phase of the ARPA-SDC Time-Sharing System will attempt to accomodate a variety of user needs in a reasonable manner. Inasmuch as the variables involved are numerous and the precise effects of various situations cannot be determined completely beforehand, it is desirable that a realistic shakedown of the system be accomplished during the course of system development and checkout. The method proposed in this note will involve a set of checkout programs which can be adjusted by system test personnel to simulate the various forms of stress which user programs will exercise upon the Time-Sharing System.

The test program system will be designed to grow as the Time-Sharing system capability expands. Thus, initial stress conditions will test the capacity of the Time-Sharing system to perform simple tasks and gradually increase the complexity of demands on the system. The test programs will be used in conjunction with the Time-Sharing system commands for full exploration of system capability.

Operational Variables

The ARPA-SDC Time-Sharing system will be expected to service a relatively large number of users in a "simultaneous" fashion. In addition, many of these users will be operating at very remote locations. The nature of user activities will vary and be reflected in an assortment of object programs. The Time-Sharing system will be required to handle, in a reasonable fashion, the gamut of user programs, both from the point of view of individual program types and numbers of "simultaneous" object programs.

The most significant general variables which must be considered in a time-sharing system are:

(1) Number of users
(2) Complexity of user operations (object programs)
(3) Capacity and capabilities of computer configuration

The third item is a relatively fixed condition, although the Q-32 configuration will be expected to expand. The first two considerations are of major importance for determining the capabilities of the ARPA-SDC Time-Sharing system.

An expansion may be performed on the user's role and the type of object programs in the system. This can provide a guide for testable system environment conditions and situations. It may also proye useful as input to the recording and accounting functions of the Time-Sharing system.

User Factors

1. No. of users (grouped by complexity of object programs)
2. Input rate (TTY)
3. Response time from computer
4. Error rates for Executive commands (Indicator of convenience of system communications)
5. Communication volume (non-program) between remote stations
6. Communication response (non-program) time between remote stations

Object Program Complexity

1. Size of program (operating form, including core data storage)
2. Operating time
3. On-line or production mode
4. Utilization of I/O (frequency, volume, and type)
   (a) High-speed channels
   (b) Low-speed channels
5. Amount of auxiliary storage required by object program for its own use.
6. Time-dependency of operation (e.g., periodic, immediate response to randomly occurring request, priority)
7. Intercommunication with other object programs
8. Source language (for debugging, updating, etc.)

The above lists are not intended to be comprehensive, but are indicative of the kind of information which reveal the magnitude of demands made upon the Time-Sharing system.

In addition to a concern over the users' operations, the operation of the Time-Sharing system will be significant in terms of:

1. Overhead time
2. Core and drum space needs of the Time-Sharing system itself.

Testing the System

During the initial developmental phase of the ARPA-SDC Time-Sharing system, the areas of interest will include only simple user operations and the effectiveness of the system in servicing these users. It is not necessary to develop a complex simulation system, such as was used in SAGE, to reproduce complete user activity. This would be a large task in itself. Instead, simple "do-nothing" routines, which run under the direct control of system test personnel, would exercise various bounds of system capacity.

The results of such testing could then be used for extrapolation of system capacity by analytic methods, as well as for improvement of the developmental system.

The test programs will be able to change their effects on the system in two ways. First, the test user can specify the version of the program to be loaded using the LOAD command. If another version is desired subsequently, a new LOAD will introduce it into the system. The second method will permit the test program to change dynamically while it is operating in the system. The desired parameters will be accepted through teletype inputs and the program will alter itself accordingly. The choice of which method can be used is dependent on the particular testing parameters involved and the existent Time-Sharing system capabilities.

The following variables for test programs are currently being considered:

1. Operating time
2. Input rate (TTY)
3. Output rate (TTY)
4. Utilization of I/O
5. Program size
6. On-line, production mode

Other variables such as priority, intercommunication with other object programs, etc., will be investigated at a later date.

Initial Test Program

The initial test module will accept teletype inputs containing instructions in the form of parameter values. These parameters will communicate to the program the length of the "do-nothing" loop. Upon completing the operation of this loop, the program will report back to the test user for further instructions.

At each teletype station, test personnel will be able to manipulate these test "object" programs for various combinations of operating times. The Time-Sharing system's effectiveness for handling these conditions can then be observed.

Subsequent test modules will add the capability for changing the amounts of teletype output to each user, the number and frequency of use of various I/O channels, and the amount of core and drum space required. This evolution will take place in step with the developmental needs of the ARPA-SDC Time-Sharing system.

TM- 1126/000/00

AUTHOR A. M. Rosenberg

TECHNICAL RELEASE J. I. Schwartz
for D. L. Drukey

DATE 3/25/63 PAGE 1 OF 36 PAGES

TECH MEMO

a working paper

System Development Corporation / 2500 Colorado Ave. / Santa Monica, California

INTERIM OPERATIONAL DESIGN AND USER'S GUIDE FOR MODEL 1 TIME-SHARING SYSTEM (TSS MODEL 1)

Introduction

The first phase of the ARPA-SDC Time-Sharing System (TSS Model 1) is expected to be available for use in the summer of 1963. At that time, the Q-32 will be utilized in the time-sharing mode for a large portion of its operations. Consequently, potential users of the Q-32 within SDC should be acquainted with time-sharing procedures and prepared with compatible object programs to run with TSS Model 1. This document is intended to provide a general overview of time-sharing operations and a guide to programming for the time-sharing environment. This document will be updated as the system development evolves and as hardware implementation affords expanded system capability.

I Concepts of Time-Sharing

The purpose of time-sharing is to make the computer more economical and accessible to increase human productivity. "Time-sharing" is not the most comprehensive term to describe what actually is involved in a time-sharing system. Several concepts are integrated into such a package, including multiprogramming, on-line man-machine interaction, space-sharing, and parallel processing. Within the limits of the computer state-of-the-art, most of these techniques have been employed abundantly in the past, but not in a generalized, comprehensive manner. The lack of an integrated, systematic approach has made on-line processing (i.e., program construction and debugging) too costly to consider for practical purposes, and has led to the use of "closed-shop" operation, with the evils of turn-around time, to maximize computer utilization efficiency. However, the concept of time-sharing permits the computer to service several users in "parallel" such that the slower human reaction time for each user may be utilized to service the others. In this manner, the computer will not be idle while waiting for a specific user to interract with his program, but is always kept busy with one of many on-line users or operating on a backlog of production jobs. The net effect is that each user will have the computer "continuously" available for his personal use.

Summary

A convenient method for system testing and checkout of the ARPA-SDC Time-Sharing will involve the use of relatively simple test "object" programs which will be able to vary their demands upon the system and thus test anticipated environmental situations.

A time-sharing system literally divides the computer's capacity among a multitude of users; computer capacity can be thought in terms of operating time and storage space. If the core memory is large enough, it may be space-shared between the current users. If it is a relatively small core memory, each user will utilize auxuliary storage (drums or disc) until his turn to operate brings his program (and data) into core. The latter method is referred to as a "swap". The operating time allocation is based on a slice of time, a "quantum", which is given to each user program in turn. User programs are operated in a queue, one quantum at a time, until the program is finished. "Finished" in the time-sharing sense means that it has run to completion (no more inputs expected), or that it has completed processing a user's input and is awaiting further input. In either case, the user can terminate his program operation at any time.

Time-sharing systems require certain basic capabilities in software and hardware. The software aspects include an Executive control system, on-line program construction, debugging and other service programs, centralized I/O operations, and an effective communication language to the system. Hardware requirements include efficient, human engineered input and output devices, memory protection and relocation registers for the space-sharing of core memory, an interrupt system for inputs, timing, errors, etc., and large amounts of main and high-speed auxiliary memory.

Time-sharing can be applied to any type of computer application, although if the area of application is too demanding (i.e., a particular program system requires too much core, operating time, and guaranteed immediate service with a high data-rate), then the computer capacity can be used up with nothing left to "share". However, a good time-sharing should be able to adjust from this worst case situation to more normal, sharable conditions. Among the applications for time-sharing which have already proven very effective, is the on-line construction, debugging and editing of programs. This approach has produced an efficient and economical way of reducing turn around time at the computer and increasing productivity. This is only one area of useful application, but one that is considered important for immediate implementation because of the need to produce other "inhabitants" for the time-sharing system. Time-sharing, however, basically can be used for any type of program operation, and will not be limited to debugging program code. Rather, the convenient "debugging" of formulation and ideas and the subsequent application of the results to useful data manipulation, reduction, and presentation is a major goal of the time-sharing concept.

Initial System Constraints

Problems always arise when a complex system is developed, particularly when design, implementation, and equipment procurement are performed concurrently. TSS Model 1 is to be ready for use by July, 1963, and this places time constraints on the effort. Many of the desirable features for the system will require more time for implementation or are dependent on hardware

availability. For these reasons, this document presents an interim descri: tion of the system under the current conditions.

The following list describes various hardware requirements which are prese: not available and the effects of their absence on system operations.

1. Additional Core Memory and High-Speed Auxiliary Memory (Disc) - Requires utilization of tapes permanent and temporary stora; Slows down accessibility of p: grams and data, affects sched: priorities, restricts number (users using tapes, restricts utilization of larger invento: of programs and data. Also restricts total number of use: and magnitude of object progr(

2. Peripheral Computer (May be available by July, 1963) - Constrains the use of select: character interrupt, limits t: size of input and output mess((TTY), limits the number of u: input channels.

3. Memory Protection (and I/O Traps) - Does not afford protection fo: the Executive system and obje(programs. Does not protect a; illegal use of I/O. Does not permit security (classified d(protection.

4. Relocation Register - Does not provide convenient m(for dynamic space-sharing re- locatability.

5. Remote Bulk Input-Output Devices (Paper Tape Readers, Printers, Etc.) - Does not facilitate large inp(or outputs at remote user sta: (e.g., for production runs).

6. Improved Typewriter Device - Teletypes are clumsy, very sl(and generally unsuitable for machine on-line intercommunic(In addition, the character se inadequate.

7. Display Consoles & Light-Pens (Expected shortly) - (Display console output gene: and light-pen inputs have not been treated in this document However, this problem area, a:

from hardware implementation, is concerned with priority aspects of scheduling.)

Since there is no guarantee as to precisely when equipment will be available for the system, the design of TSS Model 1 reflects currently available capabilities and existing constraints. As time progresses, various refinements in hardware and software will become available and appropriate documentation disseminated.

Certain functions described in this document are marked with an asterisk (*). This indicates that the function may not be available until some time after July, 1963 or not completely definable for this document. These capabilities have been identified and, to a certain extent, designed, but may not be implementable within the next few months.

II Description of TSS Model 1

The time-sharing system has been designed to service many users in a "simultaneous" manner on-line. The system can also accept production jobs on a deferred-run basis. In the worst case, the system may operate only one large program (with priority). Thus the design encompasses a complete spectrum of possible environments.

Within the time-sharing system, four levels of programs will be accepted. These are:

A. Executive System (overhead functions) - Level 1
B. General Utility Programs (service routines) - Level 2
C. Library Systems (large, special-purpose service systems) - Level 3
D. Private Object Programs - Level 4

The Level 1 programs will always remain in core memory (one bank of 16k registers). Some of the Level 2 service programs will also be kept in core in the Executive area. Level 3 may originate from drums or tape, and Level 4 will originate from tapes. A further distinction between the four levels of programs concerns the processing of communication languages. The Executive system will process completely all Executive requests for Level 1 functions. It will also partially process Level 2 (service) requests. Level 3 commands will be preceded by an indication of the service system (e.g., compilers) desired, and the further communications will then be forwarded to the service system for processing. Private object programs, once loaded and operating will also process their own inputs forwarded by the Executive system.

A. The Executive System - Level 1

The Executive System is the heart of the time-sharing system and is composed of the following elements, in addition to a central basic control sequence:

1. Interrupt Control - Responds to and determines source of operatic interrupts.

2. Teletype Interpreter - Processes Executive commands and establish appropriate action by the Executive System. This function will also perform appropriate input legality checking.

3. Scheduler - Determines which user program will be operated and wh user drum to core transfer must be initiated. Permits user program operation for a "quantur of computer time in its turn.

4. Dispatcher - Assigns user storage on drums, tapes and core. Initiates all I/O transfers.

5. I/O Package - Contains all I/O code and format conversion routi and I/O transfer routines.

6. Start-up and Clean-up - Initiates and terminates time-sharing operations.

In addition to the above named elements which are currently in production, it is expected that the following additional functions will added to the system:

7. Recording and Accounting - Records pertinent data for analysis a administration; raw recorded data is reduced to more usable forms.

8. Priority Assignment - Assigns priority indicator for user based on appropriate time and event criteria and/o by operator's command. This function will b integrated with the Scheduler.

9. Library Maintenance - Adds or updates general-purpose routines a system library. Also handles service syst library.

B. Service Programs - Level 2

Service programs will normally include utility functions applicable to on-line program debugging. They will not be initially oriented to a particular higher-order language. The following types of service functions are being considered initially:

1. Debugging Program - Will permit the examination and modification of object program code, and the controlled operation of the program for debugging.

2. Dump - Large, off-line dumps of programs and data (initially, this will be octal).

3. Tape File Maintenance (Symbolic) - Permits creation or modification of symbolic files on tape; includes add, delete, insert, print, etc..

Other Level 2 functions are under consideration for later inclusion, including text editing, timing service, trace, etc.

C. Service Systems - Level 3

Service systems are identified by language-dependent characteristics or by the magnitude or complexity of their operation. Compilers will fall into this category. At the present time, two such service systems are anticipated for inclusion in TSS Model 1. One is a special version of the JOVIAL one-pass compiler, JST, which includes a SCAMP compiler. In addition, a special program to handle simple mathematical computations, Calculator, will be developed. Other language systems such as IPL-V, LISP, and ALGOL are additional possibilities for the time-sharing system.

D. Private Object Programs - Level 4

These programs originate from individual users and will be brought in and operated upon the command of the user. The procedures for operating Level 4 programs will be discussed in the next section.

TSS Operation and Machine Error

The time-sharing operation will be initiated by the computer operator and will run until he "winds-up" the time-sharing period. If machine errors occur, it is anticipated that various startover conditions will be handled by the system. In some cases, the users may have to reinitiate their operations.

III The Initial Executive Command Functions and Language for The ARPA-SDC Time-Sharing System (TSS Model 1)

The initial phase of the ARPA-SDC Time-Sharing System will require a mi basic set of commands whereby potential users can communicate with the Executive program in the computer. It is intended that the initial set commands, called Executive Commands, will permit adequate communication facilities for the Time-Sharing control system without extending the sophistication of the system to a greater degree than necessary for the first developmental phase.

The concept of man-machine communication which will be pursued for the Time-Sharing System will provide the capability for the Executive syste to recognize and react to those commands to which it can be expected to respond. Other communication, destined for object programs and service systems, will be passed on by the Time-Sharing System to the specific u program being operated. Thus, the Executive system will not be respons for "understanding" all types of languages, and, to minimize system ove head time, the Executive system will not examine the user's communicati unless it is specifically identified as an Executive command.

The initial set of Executive system commands will be adequate for pract operation of the Time-Sharing System. However, the list is open-ended that additional commands may be added as necessary and the system respo to particular commands may be modified as the need arises.

The commands bear several characteristics which lend themselves to prac use. The words will be short in the number of characters they contain, readable, and will be hopefully unambiguous. Finally, no two commands be similar enough to make the command still appear legal if a typograph error occurs. In composing the Executive command message, the specifie format order must be followed, with at least one space between paramete fields, although excess spacing between message fields (including carri returns and line feeds) will be ignored by the Executive system.

A. The Executive Command Identifier

Executive Commands will be preceded by a special character to ident it as such. The special character has been selected for the model Teletype as the exclamation mark (!), an upper case character. The Input Interpretation portion of the Executive system will process t commands preceded by this symbol as Executive commands. If there i no such identifier, the command will be passed on to the user's ob program (or service system); if no object program exists to receive a message, the command will be rejected as illegal. The (!) can be to communicate with the Executive even while the object program is operating in order to perform various service or debugging functior

B. End-Of-Message

All teletype messages to the time-sharing Executive system must conclude with an "end-of-message" character. (This does not necessarily apply to service routine communication.) The character is the "Bell" on the A-28 teletype. Object programs will have an initial option of using the Bell character, or interrupting on every character.

C. Executive Command Response

Executive commands will be used to communicate directly with the Time-Sharing System. This includes any portion of the overhead system (level 1), common service functions (Level 2), and initiation of Level 3 and 4 programs. The commands will be examined as rapidly as possible by the control programs but response to these commands will be dependent on the nature of the request and the current user load on the system. (The scheduling algorithm will handle the queue of user requests.)

A positive response to every Executive Command will be given to the user in one of the following forms, followed by a carriage return and Bell.

(1) $ WAIT - Request being processed.

(2) $ BUSY - System operating at maximum user load, or a particular service function is currently in use and it is restricted to one user at a time.

(3) $ ILLEGAL REQUEST - Format or contents of command in error (with indication of specific error).

(4) $ READY - Command has been executed.

(5) Line Feed, Carriage Return and Bell.

The user should wait for a response from the system prior to inserting another request, unless otherwise specified. This principle applies to most on-line programs. All system printouts will be preceded by a special identifying character. This character will initially be the dollar sign ($).

D. User Identification

The Time-Sharing System will permit a variety of users to gain access to the computer. However, it is desirable that the possibility of confusion over the identity of specific users be reduced to a minimum. This is especially necessary where several users will queue up on a

single input device. For this reason, TSS Model 1 will require use identification for certain Executive commands, rather than assume t the same individual is still using a given teletype channel.

To minimize the user's identification function, an initializing LOG command will be used at the beginning of each user's operation. Th LOGIN function will accept appropriate reference information as to user's organization, man number, work order or problem number. In return, the Control System will generate a temporary user's referen number to be used for the duration of his current time-sharing oper The reference number will be used for those commands which indicate new operation being initiated by the same user. This procedure wil require the user to always "register at the front desk" prior to op ting and thus guarantee correct bookkeeping of user activity.

E. Deletion of Executive Command Entry

If an erroneous Executive command (no end-of-message character incl has been entered on the teletype, it may be deleted by entering thr (or more) line feeds in succession, followed by the end-of-message This will cause the Executive System to disregard this input, and p back to the user, "Cancelled." If the last command entry contained end-of-message character, the QUIT command or CANCEL instruction ca be used.

F. Sequence of User Commands

To initiate a time-shared operation, the user will first note any printout on the input device regarding the current status of the Time-Sharing System. Notices will be generated to all stations wh the system goes on the air, off the air, or if portions of system capability are temporarily unavailable. This information will be generated each time a user teletype channel becomes free for the n user on that channel, or when a significant change in system statu occurs.

1. LOGIN

Assuming that the Time-Sharing System is on the air, the user type in the following Executive Command:

! LOGIN ORGANIZATION CODE USER'S NAME WORK ORDER NO.
(Last and initials) or
or PROBLEM NO.
MAN NUMBER

The organization codes will be specified at a later time.

The system will respond with a busy signal (BUSY) if unable to accept a new user. Otherwise, the printout will indicate READY.

At this point, the user may enter into the system any external symbolic data or programs by creating or updating a "file" on tape. A service function (Level 2), the tape file maintenance program, will be available for this operation and is discussed later on in this section.

2. LOAD

In order to operate any binary object (Level 4) program, the following command will be used:

	User's Ref. No.	Object Program Ident.	Tape Reel No.	(End-of-Message)
! LOAD	XXX	AAAA	XXXX	BELL

The LOAD function will cause the transfer of the specified binary object program from tape to drums (if not there already) in preparation for on-line operation. If the program is not in the system (core or drums), the system will search the tape drive inventory to locate the specified reel number. If the reel number is not found, a request will be generated to the operator to load the designated tape reel onto a given tape drive. When the operator has loaded the reel and so informed the system, the program will be transferred to drums and a message sent to the user saying LOADED. If the indicated tape reel cannot be located by the operator, he will notify both the system and the user of this fact, and the LOAD request will be automatically cancelled.

If the object program will use tape or card inputs or outputs, these will be identified in the LOAD command as follows:

	Reference No. For Object Program	Tape Reel No.	(Deck Identifier)
(Inputs)	I1 XXXX	I2 XXXX I3	C AAAA, etc.
(Outputs)	O1 XXXX	O2 S O3 B O4 L O5	P, etc.

S will indicate a "scratch" tape (no particular reel number, and not to be saved after operating). B will indicate an output blank tape is needed, which must be saved. The system will then inform the user (and operator) of the tape reel no. and file name for future use. L will indicate an output tape to be listed after completion of the operation. All output tapes (O, S, B, & L) will be mounted with the file protection ring on the reel. A "C" following the Input identifier means the card reader, and a "P" following the

Output identifier means the punch. These must be available f use as indicated by the computer operator to the system. In case of the card reader, the specified deck must be loaded as

The operator procedure to load required tape reels (and card described earlier will apply to all tape needs of an object p If for any reason, the tape reels or card deck cannot be load or the I/O equipment is not available, the request will autom be cancelled, with a printout of "$ BUSY".

If the object program will communicate with more than one tel channel (multi-user interaction), the participating channels identified as follows in the LOAD command:

Ref. No.	TTY Channel No.
T1	XX T2 XX etc.

(This is not applicable to the situation where the object pro normally communicates with only one user, but several users m provide the inputs and copies of output are sent to all obser users. This capability is obtained by using the ASSIGN comm later.)

3. GO

Once the user's object program has been transferred to drums (and required tapes mounted), the system will print out "$ RE The user can then initiate its operation by the following com

```
        (EOM)
! GO    BELL
```

This instruction will cause the program to be transferred to and control branched to the program's starting location when user's turn in the operational queue arrives.

The object program will operate until the user takes a termir action, it is automatically suspended because of user inactiv (no TTY inputs when inputs are expected), or the program runs completion (does not expect input). (If a user initiates a n LOAD request on the same teletype channel, this will cause a suspension of the current program operation.)

4. STOP

The user may "halt" his operation at any time by giving the STOP command using only the ! and Bell.

```
        (EOM)
!        BELL
```

This command will retain the object program and environment in core (as long as possible) but will not operate it. At this point, the user may examine or modify portions of the data or the program he is using via Level 2 service functions and continue if desired. (See Resume below.)

5. QUIT - Q

If the user wishes to permanently cease his operation and is not concerned with preserving the program environment, he will issue the following command:

```
        (EOM)
! Q      BELL
```

The user's program and environment will be erased from drum and/or core, and the teletype channel will be made available for a new user operation.

*6. DEFER

If the user desired to temporarily suspend his operations within the time-sharing system, he may issue a command which will deactivate his program and save both program and environment for later restart from the last point operated. The command for this function is:

```
        (EOM)
! DEFER  BELL
```

This command will cause the Executive system to place the indicated program and its environmental data on a tape. The tape reel no. will be given as output to the user for future restart operation. The DEFER command should not be used for object programs using tapes or cards.

7. RESUME

Programs which have been stopped or deferred by the user can be resumed from the point of last operation by using the following command:

	Tape Reel No.	(EOM)
! RESUME	XXXX	BELL

(The tape reel no. must be given if the object program is no l in the system. This will initiate the tape load function.) I program is to be restarted from a location other than the last of operation, the new starting address (relative) may be indic in the command as follows:

	Relative Octal Address	Tape Reel No.	(EOM)
! RESUME	XXXX	XXXX	BELL

The RESUME command cannot initially be used for object program using tapes or cards. An initial restriction will be placed o user in that he may defer only one run of the same program at

*8. CHANGE

The user may make limited changes to a prior LOAD command with respecifying the entire original request. The command CHANGE permit the addition or replacement of LOAD parameters (tape re numbers and identification of input, output or teletype channe reference numbers.)

! CHANGE AAAA XXXX BELL - indicates desired binary program indicated tape reel no.

! CHANGE I1 XXXX 01 XXXX BELL - indicates revision (or add in identification of tape input a output for object program.

! CHANGE B T3 XX BELL - indicates an additional blank tape object program, and XX is an addi participating user channel for th

The CHANGE command may be issued after the user receives a "WA response from the system, but cannot be given after the GO com

9. ASSIGN

An object program, which can communicate with only one user, m observed during operation from more than one teletype station the use of the ASSIGN command. This will permit multiple copi all inputs and outputs to be sent to observer stations. (This situation does not apply to object programs which are designed communicate with several users.)

```
                  Teletype        (EOM)
                 Channel No.
! ASSIGN  XX  XX  XX              BELL
```

Inputs to the object program can come only from the teletype station originating the LOAD and ASSIGN commands. (It may be possible in the future to minimize this restriction.)

*10. CANCEL

To countermand the last Executive request inserted into the system, if appropriate, a CANCEL (Level 1) command can be given:

```
                        (EOM)
! Two Line Feeds        BELL
```

This will permit an object program to continue operation, but will cancel any Executive service which has been requested concurrently (if possible).

11. DIAL

A service which will be provided by the Executive system (Level 1) will permit the use of the teletypes for direct station-to-station communication. With the current computer configuration, the length of a teletype communication message is limited to a total of 96 characters (including all control characters such as upper and lower case shifts, line feed, carriage return, and end-of-message.) It is suggested that about 88 printable characters (including spaces) be used as a rough limit on the size of the DIAL message.

The DIAL command may be entered into the system at any time during the user's operation, as long as the system has acknowledged the last prior input. Once the DIAL input has been entered, no further input should be given by the user until the service has been completed, since that input may be partially destroyed by a new message. The DIAL function will be performed during the Executive System's overhead time.

```
          (Addressee)    Start of
            Teletype      Message
            Channel      Indicator
! DIAL       XX            Line      (Message)    BELL
                           Feed
```

Multiple addresses may be given, with an initial limit of three teletype channel addresses.

```
! DIAL  XX  XX  XX  Line  (Message)   BELL
                    Feed
```

When the DIAL message has been transmitted, the system will pri to the sender:

$ DIAL COMPLETE

12. FIND

The Executive system will provide information to the user conce the drum and core location of his programs and/or data. The FI command may be given after the object program has been loaded t system. If the program itself declares a drum storage file, FI must be used after the program has started to operate. The pri will give the assigned core and drum storage, the current queue and tapes being used by the object program.

! FIND BELL

13. RUN

Although the prime customer for time-sharing services should be programs which interact on-line with the user, it is recognized that a great deal of production work must also be processed by system. By definition, a production run will have all inputs a outputs specified prior to the operation, and no user inputs or interaction are expected during the course of the run. The pro duction run also implies that the user does not normally requi system response time necessary for on-line interaction, and tha it may be deferred until convenient for the system.

The RUN command must be instituted with all required inputs loa for system use. Thus, any symbolic data should first be "filed to tape. The RUN command will contain all the necessary inform for operating the object program; it will be preceded by the us LOGIN function.

	User's Ident (Ref. No.)	Object Program Ident	Tape Reel No.		Estimated Run Time (Minutes)
! RUN	XXX	AAAA	XXXX	Card Deck Ident	
	I1 XXXX	I2	XXXX	I3 C AAAA etc.	
	O1 XXXX	02 P	03 B etc.		XXXX

Normally production run outputs will not be transmitted to tel However, bulk output to remote locations in the form of punched paper tape or high-speed printouts may be available in the futu

Production runs may be submitted through the operator, who will enter the RUN command himself.

Production run outputs will be produced on tapes, which can then be used for printout (or possibly card or paper-tape punching), or for subsequent program input. Output tapes are identified as L tapes in the RUN and LOAD commands. These tapes will be printed out and then blanked. If they are to be printed and saved, they will be identified as normal output tapes, either O or B and the user will inform the operator about printing the tape off-line.

G. Service Commands - (Level 2)

The Executive system will process user commands which call Level 2 functions. Level 2 programs fall into two classes; Class A progra (e.g., Debugging Program) are permanently available in the Execut area of core memory, will not use tapes, and will operate on prog (or data) located in core memory or drums in small intervals of t such that the input-to-output process will be complete within one quantum of system time; Class B programs are service programs whi will use tapes and may not necessarily complete input-to-output o ation within one quantum of time. The inputs to Class B programs not have to be located in core memory prior to service operation. Furthermore, Class B functions which tie up tapes will initially one user at a time. (This will be discussed later on in this sec

In addition to the above-mentioned Class A and B programs, a third cla of routines will be available for use in constructing object programs using library subroutines.

Reversion

It is anticipated that Class A debugging routines may be employed the user intermittantly during object program operational checkout Thus, it is desirable that the process of alternately operating ar debugging be facilitated as much as possible. This will be accomp by automatic "reversion" of the user's operating status from a Lev 2 operation (debugging) to the previous operating status of his ob program. Class B routines will initially be treated similarly to Class A routines as far as reversion is concerned.

In the light of the reversion concept, it will be necessary for th user to place his object program in the status desired after servi operation prior to using the service functions. For example, if several different service operations are necessary prior to contin operation of the object program, the user should STOP his program After service, it will still be in the STOP status.

1. Class A Service Functions

a. Debugging System

Programs which have been compiled but not checked will req the convenience of a debugging tool which operates on-line within the time-sharing system. The concept of on-line de will vary from present debugging procedures. Accordingly, set of debugging routines, operating through the Executive system, will be developed for TSS Model 1. The debugging

service will allow the user to examine any address in his program (or data) in a specified form, i.e., octal, decimal, BCD, symbolic. The contents of these registers (core or drum) can then be changed. The program or data may be searched for particular values, address references, etc. Breakpoints can be inserted within the program and the program may be operated from any appropriate starting location. In addition, the contents, at any point, of the machine registers (accumulator, B register, logical register, index registers, etc.) can be obtained by the user.

A concise communication language utilizing the available keyboard character set is being developed to control the debugging package. Initially, the language it will handle comprehensively will be machine language and appropriate aspects of SCAMP and JOVIAL (JST).

b. Cost

This service function will provide the user with an accounting of his time and space utilization up to and including the final COST command.

```
                User Ident
                Ref. No.        (EOM)

!  COST         XXX             BELL
```

2. Class B Service Commands

a. Tape File Maintenance Functions

Symbolic information (programs or data) will be maintained on tape in sequential line number form. By referencing the appropriate line number, tape files can be modified and updated through the use of appropriate file commands.

Due to the fact that the initial TSS is tape-bound and tape file maintenance requires tape access, these functions will be available to only one user at a time. At the time of requesting this service, the user will be notified of three possible conditions:

(1) "$READY" - user may proceed to use the tape file service.

(2) "$RESERVED" - service program in use, but user will operate next. ("READY" will be printed

out when user can proceed.) If
does not want to wait, he may is
a "QUIT" command.

(3) "$BUSY" - service program in use and anothe
is next in line. User may try a
later.

(1) FILE - F

This command will create a new symbolic tape file, and w
require loading a blank tape by the operator.

	File Name	Tape Reel No.	Input Source
! F	AAAA	(XXXX)[1].	C = Card Reader Blank = Teletype

Inputs for FILE will be expected from the teletype channe
unless specified as C in the command. The user's identi
for LOGIN will be used to label the tape reel, and the re
number will be output to the user upon completion. The
input to the FILE routine will be an END message or card.

(2) UNFILE - FU

This command will notify the operator to blank a designat
tape file. If multiple files (belonging to one user) are
maintained on a tape reel, this command will mark the fil
for later deletion.

	File Name	Tape Reel No.	
! FU	AAAA	XXXX	BELL

(3) REFILE - FR

This command will accept a number of file updating instru
from TTY and perform the specified modifications. When
completed, it will UNFILE the old file and print out a ne
tape reel to the user.

	File Name	Tape Reel No.	
! FR	AAAA	XXXX	BELL

1. The designation of a particular tape reel number is optional.

When the system responds with a "$READY" printout, the user may insert the following inputs:

	Action Code	(Symbolic Entries)	
(a)	A	XXXXXX XXXXXX XXXXXX etc. (Line number)	- ADDS following symbolic entries to end of file
(b)	D	XXX XXX	- Deletes indicated line no. entries
(c)	I	(Line number) XXXX XXXXXX (symbolic entry)	- Inserts following symbolic entries after indicated line no.
(d)	R	(Line number) XXXX XXXXXX (symbolic entry)	- Replaces indicated line no. with following symbolic entry

Inputs to REFILE will be controlled by the service program. That is, a limit of 8 symbolic entries (not including the action code) may be transmitted at one time, and further inputs will be called for by a printout of "$READY". Entries will be separated by a line feed/carriage return. Deletion of an input can be accomplished by adding two line feeds and the end of message (BELL). This will cause the program to ignore that input. If a previous input is to be changed, a new input will override the previous one.

(4) FILE MERGE - FM

To combine two existing tape files (adding one or more to another) the FM command is used. No more than two tape reels may be involved at one time.

	New File Name	Base File Name	Tape Reel No.	Name of File to be added	Tape Reel No.	
! FM	AAAA	AAAA	XXXX	AAAA	XXXX	BELL

The resulting file will have one name for the combined elements.

(5) FILE PRINT - FP

All or portions of a symbolic file may be printed out on teletype or offline with the FP command.

	File Name	Tape Reel No.	Offline Indicator	Starting Line No.	End Line
! FP	AAAA	XXXX	(0) (optional)	XXXX	XXX

If the entire file is to be printed out, or where the rem of the file from a given starting location is to be print the letter A will be used as follows:

	File Name	Tape Reel No.	Offline Indicator		
! FP	AAAA	XXXX	(0)	A	BEL

or

			Starting Line no.		
! FP	AAAA	XXXX	(0) XXXX	A	BEL

b. Delayed Dumps

Large dumps, which are not practical for on-line teletype printout, can be specified for printout off-line. However this capability will be initially limited to octal printou

	Relative Address of Starting Location	No. of Registers to be dumped in decimal	
! DD	XXXXXX May be given in octal or symbolic tag references	XXXXX	BELL

If a drum file is to be dumped, the drum file name used by object program must be given.

	Drum File Name	Relative Start Location	No. of Registers In Decimal	
! DD	AAAA	XXXX	XXXX	BELL

Delayed dumps may be called for at any time during the user's operation. However, only one user at a time may use this service, and there may be a wait until the function is available. All delayed dumps will use the same output tape for ease of operator tape maintenance and to conserve useable drives. Each delayed dump will contain the user's identification derived from the LOGIN function.

c. Explain - EX

It is anticipated that the system be self-explanatory to the user, in that guidance or how to use the system and what it can provide may be shown to the user upon request. The EXPLAIN command will be a service function which provides "automatic" documentation on procedure and inventory availability. The initial command:

```
          (EOM)
! EX      BELL
```

Will give a printout of available explanation of different subjects, (e.g., Executive commands, service functions, Dispatcher I/O calls, etc.). The user can then select the subject code indicated and a lower level of selection will be offered. This "tree" can be followed as deeply as required (or available) by the user.

H. Service Systems Commands - Level 3

Large-scale service functions require a great deal of developmental effort, and consequently only a minimal amount of Level 3 products will be available for TSS Model 1. However, with the passage of time, other additions will be made to the system.

Service Systems will be initially called in with an Executive command, and when the system is ready for the user, all further inputs will be handled in the service system's language, until another Executive command is given. Because of the magnitude of service system operations, they will usually be available to only one user at a time. Thus, the "READY", "RESERVED" and "BUSY" printouts described for Class B service functions (Level 2) apply here.

1. JOVIAL - JTS

A version of the JOVIAL one-pass compiler will be available for the time-sharing system. It will accept symbolic inputs from tape (and cards) and produce a binary program on tape for subsequent use in

the time-sharing system. The symbolic deck should normally be on tape prior to giving the following command:

! JTS	AAAA	XXXX	D	Optional E	BELL
	Name of symbolic file = name of object program	Tape reel of input file or C = Card Reader	L	error output on teletype	
			N		

D = Detailed listing - JOVIAL and
L = Straight listing (JOVIAL)
N = No listing

JTS will not require any further control inputs other than the initial command. If cards are used, the operation will be hand by the operator, as described for LOAD.
When the compilation is complete, a printout will indicate thi along with the tape reel no. of the binary output.

JTS can be operated in the production (deferred operation) mode naming the JTS as the object program in the RUN command. The tape reel number, in this case, will be omitted.

The detailed operation of JTS will be described in other docume

2. SCAMP - (JTS)

A version of SCAMP for the time-sharing system will be availabl via the JOVIAL compiler. The symbolic input file will contain appropriate indication that direct code will follow, and it wil be compiled properly. The same Executive command and parameter for JTS is applicable. Further information on the compiler languages will be documented shortly.

*3. Calculator - CALC.

A program which will perform simple arithmetic computations wil be developed for the time-sharing system. It will operate in a manner similar to a desk calculator. At present, the calling commands cannot be specified, but will be documented in the fut (Calculator may be a Level 2 function, residing permanently in core.)

III. Programming for TSS Model 1

This section will deal with the manner in which object programs for the time-sharing system will be designed and coded. It is not possible to furnish all the details at this time, but as they become available, they will be appropriately documented.

A. Concept of Time-Sharing Programs

Programs which will operate on-line in a time-sharing system should possess several inherent characteristics. They should hopefully be small, fast and operate on minimal inputs, they should minimally involve the use of tapes, and they should be capable of communicating with the user via the teletype. In addition, all I/O functions will be performed by the Executive system, and thus a centralized I/O package will be utilized by all object programs.

Undoubtedly, object programs cannot all be small, since the magnitude of problems which can benefit from time-sharing may be large. Programs large and small can always be run in a production mode (RUN). However, one must remember that there is always an upper limit on the capacity of the system which could be completely used by one object program, leaving nothing to share in a practical sense. The time-sharing system will be geared to handle a worst case, with appropriate limitations on additional users.

Programs which are to interact on-line with a user should optimally reply to the user's input in accordance with the user's response time needs. Furthermore, since this is a two-way communication situation, the program should provide appropriate positive responses to each logical user input. That is, the object program should control the user's input rate, if necessary, by providing pertinent feedback for more input.

Object programs, which will be generally useful for service to several independent users, should be designed as "pure" programs. Conceptually, "pure" programs do not modify themselves in any way, and all specific user references are maintained in associated tables. With this technique, programs may be interrupted at any point for a given user and resume service for that user at a later time. This will permit one program to service several users without replication for each one. Generally speaking, however, programs of this nature will probably not be Level 4 types, and will be part of the service system.

B. Initial Priority Considerations for Time-Shared Scheduling

In TSS Model 1, the scheduling function will discriminate between small and large programs and programs which use low-speed I/O and

those which do not. The three categories which exist are listed i the order of precedence for system:

1. No low speed I/O (card reader, punch, tapes), and size less than 16k (including table space)

2. No low speed I/O and size between 16k and 32k

   or

   Low speed I/O and size less than 32k

3. Program size exceeds 32k

The Scheduler will determine the priority of a program from the LO command which indicates input and output needs (i.e., I1, I2, O1, L, etc.). Programs of higher precedence will be serviced prior to those of lower precedence.

Production Runs

Programs which are initiated with the RUN command will be operated only when no on-line programs require service (i.e., waiting for input). When they are operated, they will run to completion (or estimated running time) without quantum interrupts, unless on-line users interrupt for service. Programs which are checked out and available for production runs will be maintained on a production library tape. Level 3 service systems may also be used for produc purposes by giving the RUN command and identifying the service sys (e.g., JTS) as the object program.

C. Program Identification

Programs to be operated in TSS Model 1 will have their identifying tag names limited inftially to four alphanumeric characters, not including modification numbers. Mod numbers may not be necessary for program maintenance in the system, since only one valid versio of a particular program should exist. If versions differ function ally, then they should have a variation in the name. However, the is no restriction on how the identifying characters are used. (Sy programs may adopt a unique identification character in the future which may not be used by object programs).

The reason for limiting the ident to four characters is because of equipment constraints; for reading a tape by ident, only four characters are used.

D. Programming Languages and Compilers for TSS Model 1

Any binary program compiled outside of the TSS may be used in it provided that it has been compiled with addresses relative to zero, and it uses the Dispatcher calls for all its I/O activities. In addition, the program must be loaded by a special program into the binary tape format used by the system. The current J-2 compiler for the Q-32 may be used in this way, if necessary. Therefore, to utilize an existing program in the TSS, the I/O references must be changed to conform with Dispatcher calls, and the modified binary program loaded on to tape.

A special version of the JOVIAL one-pass compiler is being written for TSS Model 1 which will generate a binary tape suitable for use by the system. The name of this compiler is JST, and it will accept card or tape input. In addition, JST will be capable of compiling SCAMP code, either as part of a JOVIAL program or entirely in SCAMP. In the latter case, the SCAMP code must be enclosed in the normal JOVIAL direct code brackets. The use of JST will be documented in detail shortly. A current reference which is pertinent is TM-970/002/00, JOVIAL-X.2, The Language of The One-Pass JOVIAL Compiler.

One problem which exists is the incompatibility of Hollerith constants between J-2 and the present SCAMP. SCAMP does not translate into sortable Hollerith whereas J-2 does. As a result, it will be necessary for the Dispatcher to be aware of the fact that sortable Hollerith is being used and consideration is being given to have JTS handle either mode. A procedure involving the Dispatcher calls is planned whereby the use of sortable or unsortable Hollerith can be indicated to the system.

E. Program Size and Data Tables

Programs will have to make provision for the amount of storage space required for the program itself and the maximum amount of data it can expect. Extra storage space can be reserved by declaring dummy tables. Programs may declare (via file declarations) required storage space on drums in addition to the core space used while operating. The object program will initiate all transfers to and from such drum space via the Dispatcher, during its quanta of operating time. Program "systems" may thus have its own monitor routine which operates different program subsets on an input-dependent basis or in a given sequence.

F. Programming for Teletype Communication

On-line programs for the time-sharing system will intercommunicate with the user via teletypes initially. The use of display scopes and light pens will be treated in future documentation. Until a peripheral computer becomes available, the input memory and output buffer drum will be used for in-out teletype activity. These place certain constraints on the object programs in the system.

1. INPUT

Input memory space will be allocated among the available teletype channels, resulting in a maximum figure of 160 characters (20 words) for the total number of characters (including all control and non-printable characters) which may be entered by the user at one time. The current means of "transmitting" the input to the system is either via an end-of-message (Bell) character, which will generate an operational interrupt in the computer, or the object program can specify an interrupt on every character. This will cause the Executive to convert and examine the teletype code. If the message is for an object program, it will be stored in input memory until the object program operates. At that time, the message will be transferred to the designated "File" area in the object program's core space. (For character interrupt, all newly arrived (and converted) characters will be transferred.) Normally, the user should wait for a positive response from the object program prior to entering another message, unless otherwise specified.

The object program when ready for a new input message, must make a call to the Dispatcher. The normal procedure for teletype interraction is for the object program to generate an output and wait for an input. For this purpose, there may be a convenient method for an I/O call to combine a teletype output call with a subsequence input call, eliminating the need to return to the object program in between.

Another feature is the "no-wait" option. In this case, an I/O call for teletype input may be made, but the object program will continue to operate. When an input from the teletype arrives, it will be placed in the "file" area of the program before the next quantum of operating time. It is not advisable for the object program to use the same "file" area for both input and output messages, especially in the light of the above procedure.

Procedures must be established by object programs to cancel or delete a current input or previous input. The Executive system

currently uses three or more line feeds for this, and it may be useful if all teletype characters are normally valid input.

2. OUTPUT

Teletype output messages may be generated at any time by the object program, using an I/O call to the Dispatcher. The program will not operate until the message has been output to the teletype. The maximum size of an output message is 12 words or 96 characters, including all control, non-printable characters. Shorter messages should terminate with an end-of-message (Bell). For the case of the maximum size output message, the waiting time will be at least 16 seconds before the object program can operate again.

An object program using the teletype should consider initializing the teletype for the user prior to his next input. This normally consists of giving a line feed, carriage return, and BELL. Unfortunately, this will leave the machine in the upper case, but the end-of-message (Bell), which must be last, is upper case. This may make things inconvenient for the user to first go to lower case before typing in an input. (It is fine for Executive commands which start with an upper case character (!).)

3. General

The Dispatcher calls consist of loading the accumulator with the absolute address of the table containing the calling parameters and branching to an absolute address in the Dispatcher. This means going into direct code when using JOVIAL. The instructions will consist of the following for J2:

```
    DIRECT
Assign A = PLACE $          (PLACE is an arbitrary item which
BUC          312'           has been set to the LOC of the
                            calling table.)
    JOVIAL
```

For the new JST compiler, the references will be easier.

```
    DIRECT
LDA          CALL           (CALL is an arbitrary table name
                            where the I/O calling parameters
BUC          312'           are contained.)
```

The above procedure is also applicable for file declarations. Eventually this function may be incorporated into more convenient

library procedures. The actual parameters are described in section entitled, The Use of I/O.

4. Future Developments

When the peripheral computer becomes available, many of the straints indicated above will be removed. In addition, sin character interrupt will eliminate the need for an end-of-m (Bell) unless so desired. It is anticipated that an object gram may initialize the peripheral computer to interrupt on specific set of characters or all characters by means of an initializing call to the system.

If other input devices become available, the character set change with appropriate implications for the input and outp processing conventions. Also, the general needs of object grams for input scanning and output message composition may serviced by convenient library procedures.

G. Normal Exits of Object Programs

Programs which will operate under time-sharing should never com a halt. Two exits will be available; the first exit is to the Executive system, indicating complete termination of its operat the second exit is to the Dispatcher for new teletype input. I addition, programs will exit to the Dispatcher whenever any I/O action is required.

If an object program (on-line) hangs-up in a loop, the user sho be aware of any excessive response time and stop the program fo analysis. A halt in the program will generate an error interru this will cause an error message to the user. Production runs be operated according to the estimated running time indicated b the user. When that time elapses, the operator will be notifie and the program discontinued. (The procedure may allow operato control to continue the run.)

The exit addresses to the Dispatcher and Executive will be publ at a later date.

H. TAPES

1. Tape Formats

In general, tape inputs used by an object program are not re stricted to any particular format. However, large tape rec should not be used, so that data transfers can operate with smaller units of I/O time for time-shared, on-line operatio

2. Tape Reel Maintenance

Tape reel maintenance will be a problem for the system and not conveniently solved. For the time being, the following tape utilization procedures will be followed:

Levels 1, 2, 3

System Programs (Level 1, 2, & 3) - will be on special tapes

Level 4

Symbolic Tape Files - will be normally maintained on a single tape for each user, unless the user desires his files on separate tapes. Updated files will be added to the end of tape and obsolete file marked for deletion. (At some time, tape files will be cleaned up.)

Binary Programs - unchecked-out - will be placed on individual tape reels.

* Binary Programs - checked-out - will be loaded on a community tape unless specified otherwise. The procedure for this operation will be defined some time in the future. The programs on this tape can be used for production or on-line operation.

* Deferred Programs - will be loaded on separate tapes initially.

I. The Use of I/O

All object programs will be required to utilize the centralized I/O package provided by the system. This will be accomplished by using calls to the Dispatcher indicating the particular I/O service required. The I/O package will take care of any necessary code conversion or equipment formatting.

Prior to asking for an I/O transfer within an object program, the program must first declare a File for the use of that type of I/O activity. Then, when appropriate in the program, an I/O call is made to the Dispatcher referencing the file area (a table). The file and I/O call formats have been designed in a very flexible manner, readily expandable for additional I/O functions. Except for the first control word containing the object program name and number of words following, the word elements of the calls may be arranged in any order.

The following table lists the current formats for Dispatcher calls. These are documented on card formats (on tape) and will be updated as necessary.

LCC 3-7-63 8 DISPATCHER CALLS

1 FORMAT OF CALLS TO THE DISPATCHER

1.1 CALLING SEQUENCE CONVENTIONS

WHEN A PROGRAM MAKES A CALL TO THE DISPATCHER IT WILL LE IN THE ACCUMULATOR THE LOCATION OF ITS PARAMETER REQUEST TABLE.

THE DISPATCHER WILL SAVE APPROPRIATE REGISTERS AND RESTORE THEM FOR SYSTEM PROGRAMS, STORE THEM FOR LATER RESTORATION FOR NON-SYSTEM PROGRAMS.

1.2 CALLING PARAMETER TABLE FORMAT

1.2.1 THE FIRST WORD IN THE TABLE IS OF THE FOLLOWING FORMAT.
BYTE 0-3 NAME OF CALLING PROGRAM 1 TO 4 CHARACTERS, LEFT JUSTIFIED
BYTE 7 NUMBER OF WORDS IN THIS TABLE, NOT INCLUDING THIS WORD

1.2.2 THE INFORMATION IN THE MAIN BODY OF THE TABLE IS OF THE FOLLOWING FORMAT.
BITS 0-35 PARAMETER NAME- 1 TO 6 CHARACTERS, LEFT JUSTIFI
BITS 36-41 VALUE TYPE- INTEGER WITH THE FOLLOWING VALUES
0=VALUE IS INTEGER 42 47 OF THIS WORD
1=VALUE IS INTEGER IN NEXT WORD, RIGHT JUSTIFIED
3=VALUE IS FLOATING POINT NUMBER IN NEXT WORD
4=THERE IS NO VALUE WITH THIS PARAMETER NAME
5=VALUE IS 1 HOLLERITH CHARACTER BYTE 7 THIS WORD
6=VALUE IS 2-8 HOLLERITH CHARACTERS LEFT JUSTIFIED IN NEXT WORD. NUMBER OF CHARACTERS IS IN BYTE 7 THIS WORD (A BINARY INTEGER).
7=VALUE IS AN ARBITRARY ARRANGEMENT OF INFORMATION I NEXT N WORDS. N IS A BINARY INTEGER IN BYTE 7 OF T WORD

1.3 IN ANY REQUEST THERE CAN BE ONLY ONE OF THE PARAMETER NAM PRECEEDED BY A * BELOW.
THE ORDER OF THE PARAMETER NAMES IN THE CALL IS NOT SIGNIFICANT.
THE PARAMETER NAMES ARE ALL OPERATED ON AS 6 CHARACTER ITEMS, LEFT JUSTIFIED BLANK CHARACTERS, NOT ZEROES MUST FILL IN UNUSED CHARACTERS

2.0 THE FOLLOWING LIST DEFINES THE VARIOUS REQUSTS TO THE DISPATCHER AND THEIR FUNCTION
2.1 FILE - THIS IS USED TO RESERVE I/O EQUIPMENT FOR PROGRAMS AND TO MINIMIZE TRANSMISSION OF INFORMATION ON MOVE CALLS. IT ALSO MAY RESERVE A BLOCK OF SPACE ON DRUMS. ALL FILE DECLARATIONS SHOULD BE MADE AT THE BEGINNING OF A PROGRAMS OPERATION.
2.2 DEFILE - CANCELS ALL RESERVATIONS AND INFORMATION STORAGE CAUSED BY FILE
2.3 TAPMUV - USED TO MANIPULATE TAPE, BUT NOT FOR READING TAPE.
2.5 ASSIGN - ASSOCIATES FILE NAME WITH LOGICAL I/O UNIT
2.6 MOVE - CAUSES INFORMATION TO BE TRANSFERRED TO OR FROM CORE ACCORDING TO INSTRUCTIONS IN FILE DECLARATION
2.7 SCROLL - CAUSES INFORMATION WHICH HAS BEEN PUT ON DRUMS BY OBJECT PROGRAM TO BE ENTERED INTO INVENTORY
2.8 TOCORE - CAUSES ENTTIES IN INVENTORY TO BE BROUGHT TO CORE
2.9 TODRUM - CAUSES INVENTORIED ENTITY IN CORE TO BE PLACED ON DRUM
3.0 DELETE - CAUSES PROGRAM OR TABLE TO BE REMOVED FROM INVENTORY
3.1 LOAD - CAUSES PROGRAM ON INDICATED BINARY TAPE TO BE RELOCATED AND PUT ON DRUM
3.2 DIRECT - CALL CONTAINS 3 WORD CALL TO I/O PACKAGE IN I/O PACKAGE FORMAT
3.3 CON CALL FOR USE OF SCHEDULER TO DETECT CORE CONFLICTS
3.4 IOCINT - ENTRY FOR IO COMPLETE INTERRUPT

PARAMETER NAME	COMMENT NUMBER	TYPE	SIZE	VALUE
4.1 * FILE		6	6	NAME OF FILE
UNIT		0		0=CARDREADER
				1=PRINTER
				2=PUNCH
				3=TAPE
				4=DRUM
				5=INPUT MEMORY
				6=I/O REGISTER
				7=CONSOLE TYPWRITER
				8=TELETYPE
FORM	(1)	5		B=BINARY
				H=HOLLERITH
				C=7090 CORE HOLLERITH
INLOC		A) 1		LOCATION OF FIRST CORE REGISTER TO TRANSFER-BINARY
		B) 6	6	NAME OF PROGRAM OR TABLE IN SYSTEM

	DRMLOC	(2)	A) 7	1	BITS 18-23 0=DRUM A 1=DRUM B 4=DATOR DRU BITS 30-34 = DRUM FIELD BITS 35-47 = DRUM ADDRES
			B) 6	6	NAME IN INVENTORY
			C) 0		0=RESERVE DRUM REGISTERS FILE NAME EQUAL TO NUMW
			D) 1		RESERVE DRUM REGISTERS FILE NAME EQUAL TO INTE IN NEXT WORD
	DENSTY	(3)	5		H=HIGH L=LOW
	NUMWDS	(4)	1		NUMBER OF WORDS TO BE MOV
	TAPE	(5)	0		LOGICAL TAPE NUMBER

COMMENTS

(1) NOT USED IF UNIT EQUALS DRUM, INPUT MEMORY
(2) USED ONLY WITH DRUM
(3) USED ONLY WITH TAPE
(4) NOT NEEDED IF DRMLOC VALUE TYPE EQUALS 6 OR 0 OR IF NUMWDS PROVIDED IN MOVE REQUEST
(5) OPTIONAL FOR USE WITH TAPE

PARAMETER NAME	COMMENT NUMBER	VALUE TYPE	SIZE	VALUE
* DEFILE	4.2	6	6	NAME OF FILE TO BE EXPUN
* TAPMUV	4.3	6	6	FILE NAME IDENTIFYING LOGICAL TAPE

	OPTION		0		4=WRITE END OF FILE	237	3
					5=WRITE END OF TAPE	238	
					6=REWIND	239	
					7=BACKSPACE A RECORD	240	
					8=BACKSPACE A FILE	241	
					9=SET HIGH DENSITY	242	
					10=SET LOW DENSITY	243	
4.5	* ASSIGN		6	6	FILE NAME	244	5
	TAPE		A. 0		LOGICAL TAPE NUMBER	245	3
			B) 4		IF VALUE TYPE EQ 4 DISPATCH-	2452	1
					ER ASSIGNS TAPE AND LEAVES	2454	
					LOGICAL TAPE NUM IN VALUE	2456	
	PNAME		6	6	PROGRAM NAME OF PROG USING	2458	3
					FILE IF NOT SAME AS CALLING	24585	
					PROGRAM	24586	
	TTY		0		TELETYPE NUMBER	246	3
4.6	* MOVE		6	6	FILE NAME	247	5
			0		RELATIVE LOCATION IN SYSTEM	248	1
					TABLES OF ENTITY TO BE	249	
					MOVED	250	
	IDENT	(1)	6	4	IDENT CHARACTERS FOR TAPE	251	
					TRANSFERS CAUSES THIS	252	
					READ OR WRITE TO BE IN	253	
					IDENT MODE	254	
	NUMWDS	(3)	1		NUMBER OF WORDS TO BE MOVED	2542	3
	INPUT	(2)	4		CAUSES TRANSFER TO CORE	255	3

	COREIX (1)	1		NUMBER TO BE ADDED TO IN OF FILE DECLARATION TO ESTABLISH MODIFED INLOC THIS TRANSFER ONLY
	DRUMIX (1)	1		NUMBER TO BE ADDED TO DR REGISTER AS DEFINED OR REQUESTED IN DRMLOC OF F DECLARATION
	TNSTAT	0		THIS ITEM SET BY DISPATCH AT COMPLETION OF READ IF PROVIDED BY CALLING PROGR 0=NORMAL 1=END OF FILE 2=END OF TAPE
	WDSIN	1		DISPATCHER INSERTS NUMBE WORDS READ FROM TAPE IF PROVIDED

COMMENTS

(1) OPTIONAL
(2) EITHER INPUT OR OUTPUT BUT NOT BOTH- ARE USED
(3) NOT NEEDED IF GIVEN IN FILE DECLARATION, IF USED OVERRIDES FILE

4.7	* SCROLL		6	6	FILE NAME IDENTIFIS AREA NOW ON DRUMS TO BE GIVEN INDEPENDENT IDENTITY
	NAME		6	6	NEW NAME OF ENTITY
4.8	* TOCORE		6	6	NAME OF PROGRAM OR TABLE

4.9 * TODRUM	6	6	NAME OF PROGRAM OR TABLE	270	5
5.0 * DELETE	6	6	NAME IN INVENTORY	271	5
5.1 * LOAD	1		CHANNEL IN PQU OF PROGRAM TO BE LOADED	271 272 273	5
5.2 * DIRECT	7	3	NEXT THREE WORDS CONTAIN I/O CALL IN FORMAT OF I/O PACKAGE. THERE WILL BE LEGALITY CHECKS ON THIS REQUEST. USE ONLY AFTER CHECKING WITH SYSTEM DESIGNERS.	275 276 277 278 279 280 281	5
5.3 * CON	1		INTEGER IS CHANNEL NUMBER IN QUEUE OF PROGRAM TO BE CHECKED FOR CONFLICTS. DISPATCHER INSERTS IN BYTE OF WORD CONTAINING CON 0=NO CONFLICT 1=PGM(S) IN CONFLICT BUT CAN NOW BE MOVED 2=CONFLICT PROGRAM(S) NOW DOING I/O 3=CONFLICT WITH PROGRAM IN PCUR	282 283 284 7285 286 287 288 289 290 291 2911 2912	5
5.4 * IOCINT	4		THIS INDICATES AN IO COMPLETE INTERRUPT USED ONLY BY SYSTEM	292 293 294	5

FIN

This document was produced by SDC in performance of contract SD-97

6/27

TECH MEMO

a working paper

System Development Corporation / 2500 Colorado Ave. / Santa Monica, California

TM- 1126/004/

AUTHOR C. E. Fox

TECHNICAL

RELEASE J. I. Schwa

for D. L. Druke

DATE 6/27/63 PAGE 1

INITIAL TIME-SHARING SYSTEM DEBUGGING CAPABILITY

TABLE OF CONTENTS

Introduction

Included in the description of functional capabilities of TSS Model 1 (TM-1126/000/00) is on-line debugging. This debugging capability refers to a set of permanently available routines in the Executive System that will perform the system user's requests for on-line debugging. An initial version of the debugging routines (hereafter referred to as program DBUG) are now being checked out with TSS Model 1. The purpose of this paper is to provide a brief functional description and user's guide of program DBUG.

1.0 Functions of DBUG

The requirements of on-line debugging are based on the programmer's need to examine parts of his program, to control its execution, and to change the program during check-out. The following functions of DBUG provide a way of meeting these requirements.

1.1 Displaying a Register

The contents of any memory register may be examined by teletyping a debugging request to open that register.

1.2 Changing a Register

The contents of a register may be changed by teletyping a new value to be inserted.

1.3 Inserting Breakpoints

Breakpoints may be inserted into an object program so that various kinds of debugging can be performed when the program reaches such breakpoints.

1.4 Dumping Blocks of Registers

A block of consecutive registers may be printed on the teletype in octal format by inputting a starting address and a count.

2.0 Debugging Commands - Language Conventions

The special character exclamation point (!) is used to identify TSS system command inputs. The special character quotes (") identifies an object program input. The first debugging command must be preceded by an exclamation point when changing from object program type of inputs t system command inputs.

The following outlines the remaining general requirements of debugging inputs.

2.1 Control Characters

Each debugging command must terminate with onr or more of the thr teletype characters: slash (/), number sign (#), and Bell (end-of-message key).

Slash (/) followed by Bell terminates a command to open a registe

Number sign (#) followed by Bell terminates a command to change a register.

Bell (end-of-message key) terminates a command to specify a mask or format change.

2.2 Blanks

In general, blanks are ignored by DBUG. A blank (space bar on teletype keyboard) only serves as a field delimiter except when input as Hollerith value (enclosed in parentheses). Wherever one blank may occur, any number of blanks may occur.

2.3 Numeric Inputs

Numbers used in debugging commands may be signed or unsigned octa decimal, or floating point; they may contain the following charac

0 1 2 3 4 5 6 7 8 9 . E + − '

Octal numbers are defined as being any string of digits 0 through 7, terminated by an apostrophe (').

Decimal numbers are defined as being any string of digits 0 through 9, terminated by a blank or any other special character (#, /, +, etc.) except apostrophe (') or period (.).

Floating point numbers are defined as being any number containing a period (.) and/or terminating with an exponent. The value of the exponent is the power of ten by which the number is to be multiplied. The exponent starts with the letter E followed immediately by the power of ten, which may be signed or unsigned. The absolute value of the exponent must be less than 303.
Example: 0.25E-6.

2.4 Symbolic Name Inputs

Any string of six or less alphanumeric characters beginning with a letter and terminated by a blank or any special character is considered a symbolic name.

2.5 Hollerith Inputs

Hollerith values may be used in debugging commands by bracketing the value with parentheses. Prefacing the left parenthesis is the count of the number of characters and the type. Type can be Q-32 machine Hollerith (non-sortable) indicated by letter H, or 7090 core Hollerith (sortable) indicated by letter C.
Example: 8H(A/.(0ØW#).

through
, +,

ining
of

immedi-
The

with a
con-

ng
the
Q-32
090

3.0 Operating Instructions

The following pages contain the detailed description of how the vari functions of the debugging routines are used. However, certain cond: tions must be met before requests to operate the debugging routines would be allowed.

a. An object program must have previously been loaded by the System Load Command.

b. The execution of the object program must not have been terminate by the System Quit Command.

3.1 Opening Registers

In order to open a register and have the contents printed on the teletype, the address or a relative address followed by a slash must be input. Once the address of a register is specified, it remains the open register address until a different one is specified. The following examples will illustrate opening a register.

```
40000'/   (octal)
16384/    (decimal)
```

Teletyping either of the above commands will cause DBUG to print the contents of octal address 40000' on the teletype. DBUG will print the address and a sixteen character octal representation of the open register.

3.2 Stepping the Open-Register Address

```
+1/
-100'/
```

The above inputs will change the open register address by the amount shown (e.g., 40000' becomes 40001' in the first example) and the contents will then be printed.

```
+0/
-0/
/
```

The latter three examples all have the same effect, that is, to reopen the previously opened address.

3.3 Dumping

4440' 10/

This input will cause an octal dump to be printed. The first number is the starting address of the dump and the second is the number of registers to dump. After the dump, the last register printed (4451') will then be the open register address. The maximum number of registers allowed per dump is 20.

3.4 Opening Machine Registers

$A/	Accumulator
$B/	B Register
$L/	Logical Register
$P/	Program Register
$X1/	Index Register 1
$X2/	Index Register 2
$X3/	Index Register 3
$X4/	Index Register 4
$X5/	Index Register 5
$X6/	Index Register 6
$X7/	Index Register 7
$X8/	Index Register 8
$LIV/	Live Register
$D/	Double Index Selection Register

Any of the above inputs will cause the contents of the corresponding machine registers to be printed. The settings of these registers will be as they were when the execution of the object program was last interrupted. The environment at that time is saved, along with other information used by the system in a block of registers at the end of the object program. These environment registers, therefore, are the ones that are displayed in the examples shown above.

Also, the address in the environment block of the machine registe: referenced then becomes the open register address.

3.5 Displaying the Panel

$ PANEL/

This debugging command will cause all the machine environment registers referred to in section 3.4 to be printed. Currently this is done by printing 14 lines of output, one register per line.

3.6 Changing Register Contents

In order to change the contents of any register in the object program (instructions, data, or environment area), it is first necessary to open that register. The value to be inserted foll by the number sign (#) can then be input. DBUG will indicate the change is effected by typing the response $IN. The value inserted will replace the entire contents of the open register ur a mask was previously specified indicating thay only part of th register be changed. Specifying masks is described in section The following example shows the method of changing a register:

```
40001'/                                  Debugging Command
C(040001') = 111111111111111'            Debugging Response
0#                                       Debugging Command
$IN                                      Debugging Response
```

This sequence of commands to and responses from DBUG would firs open register 40001' and then set it equal to zero.

The input value may be an octal number, a decimal number, or a floating point number. The number is always right justified within the active bits of the open register. (All bits are active unless otherwise specified by a mask.)

Note: The Live Register ($LIV) contains the address at which t next operation of the program will commence. Therefore, one may alter the sequence of his program by changing the Live Register.

3.7 Setting Masks

When only certain bits of the open registers need to be changed, it will sometimes be easier to specify a mask that will apply to the changes that will be sbusequently input. This is done by specifying the starting bit position of the mask and the number of bits in the mask. DBUG will respond by printing the octal representation of the mask. Once a mask is specified it remains in effect for all change inputs until a new mask is specified. The following examples show how masking a change can be done.

```
40004'/                                  Debugging Command
C(040004') = 2000037616000020'           Debugging Response
18  6                                    Debugging Command
MASK = 0000007700000000                  Debugging Response
60'#                                     Debugging Command
$IN                                      Debugging Response
/                                        Debugging Command
C(040004') = 2000036016000020'           Debugging Response
```

anged,
y to
y
umber of
repre-
in
The

4.0 Error Messages From DBUG

A variety of errors can occur while inputting debugging requests. Wh an error is encountered, DBUG prints an error number and the request ignored. The following is a list of the error numbers and what they mean.

ERR 1	Inactive teletype
ERR 3	Program not in core
ERR 6	Load in progress, a system error
ERR 7	Illegal input, should be easily recognized error in the context of the preceding debugging request
ERR 8	Octal number containing an 8 or 9 digit
ERR 9	Numeric input too big, 14 digit decimal, 16 digit octal
ERR 10	System error, this input may be later recognizable when a new function is added
ERR 11	Too much input, more than 32 characters in input message
ERR 12	Floating point number format error
ERR 13	Symbolic name greater than 6 characters
ERR 14	Opened address greater than 177777'
ERR 15	Attempt to change a register outside of object program re
ERR 16	Illegal request for a panel register
ERR 17	No such index register
ERR 18	Opened address is zero or minus
ERR 19	Insertion format error
ERR 20	Too large a dump request, maximum of 20
ERR 21	Illegal symbolic name
ERR 23	Should be only one exclamation point
ERR 24	Illegal control character, should be / or #
ERR 25	Illegal mask, too big or too small

7/22

TECH MEMO

a working paper

System Development Corporation / 2500 Colorado Ave. / Santa Monica, California

TM- 1126/00

AUTHOR A. M. Rosen

TECHNICAL

RELEASE J. I. Schw

for D. L. Drul

DATE 7/19/63 PAG

This document supersedes TM-1126/0 dated 5/15/63, TM-1126/002/01 date 7/3/63, TM-1126/003/00 dated 6/6/6 all modifications, N-20427 dated 7

ADDENDUM TO TIME-SHARING SYSTEM (TSS-1)

USER'S GUIDE (TM-1126/000/00)

Introduction

TM-1126/000/00, Interim Operational Design and User's Guide for Model 1 Time-Sharing System (TSS Model 1), presented a comprehensive overview of the current system being developed for July, 1963. During the course of system implementation several operational design changes have been made, as well as certain system conventions. Furthermore, certain concepts in the original document were not adequately explained. This addendum describes present operational changes in the time-sharing system and clarifies certain time-sharing system concepts and programming techniques.

1. Quantum Interrupt

In the general discussion of time-sharing, reference was made to the on-line operation of object programs on a quantum basis. The quantum is a small slice of time, during which an object program can operate. An object program will run for one quantum and if it is not finished (i.e., it has not returned control to the Executive system at its logical conclusion or because of I/O transfer requests), an external timer will generate an interrupt, the quantum interrupt. The object program will be frozen at that point until its turn to operate occurs again; at that time, it will be given operating time again.

The scheduling algorithm does not always limit object program operation to just one quantum each time. A Category 1 program (programs less than 16K, which do not use low-speed I/O (tapes) will begin with one quantum, and if not finished, will double the quantum length each successive time its turn to operate occurs (i.e., 2 quanta, 4 quanta, etc., to a maximum of 8 quanta). Category 2 programs (programs which use low-speed I/O and range in size from 16K to 32K) will start out with 2 quanta of operating time and double each successive turn to a maximum of 8 quanta. Category 3 programs (programs over 32K) will always operate on an 8 quanta basis.

Production runs (RUN) will not operate on a quantum basis, but will run until complete or interrupted by a higher-level program (Category 1, 2, or 3).

2. Executive and Object Program Message Switch

The Exclamation Mark (!) used to identify an Executive command will not have to be repeated for consecutive Executive commands. That is, if several Executive commands follow each other directly, only the first one must be preceded by the Exclamation Mark. To send inputs to an object program, an initial Quotes symbol (") must be given, at the start of a new* message, and all further input will be passed on to the object program. To revert back to the Executive mode, the Exclamation Mark (!) is entered again. This procedure provides a simple two-way switch whereby the user may rapidly change between the Executive and object program mode of input.

3. "Pure" Programs

Some confusion resulted from the reference to "pure" programs. A "pure" program is considered to be one that does not modify itself and contains all object data (user context) in tables. This approach would permit a program to service several users in parallel; that is, while processing one user, it can be interrupted, service another user, interrupt, continue processing the first user, etc. The "pure" program separates context from the computational processes. There is no need for object programs to be concerned about being "pure"; such a technique is primarily applicable to general system service functions.

4. Using JOVIAL J-2 For TSS-1

The currently available J-2 JOVIAL can be presently used to program for TSS-1. The only change required is that all I/O references utilize the Dispatcher I/O call format. In addition, the program should not come to a halt (STOP); this instruction should be replaced with a

BUC	303'	
Col. 8-10	Cols. 24-27	- An item may be equated to this absolute address.

The J-2 symbolic deck should be compiled without a COMPILE card. (Begin with START to eliminate the dictionary.) The binary deck must then be loaded on to tape with a special tape-loading program of the time-sharing system. In JTS, a GOTO SYSTEM $ will do the same as above. No J-2 library procedures are valid for time-sharing except LOC.

*After GO and RESUME, the (") mode is automatically entered by the system.

5. Dispatcher I/O Calls

The calling sequence for I/O in TSS-1 has been designed to be very flexible in terms of format and expandability for new I/O functions. (See Appendix A.) Several useful hints can be given as to their use object programs.

A. The I/O Call Format

The I/O calls are basically Hollerith tables with some non-Holler entries. The order of the parameter entries is not fixed except the first control word and certain two-word combinations. The fi control word must always contain the following:

Bytes 0-3 — Name of object program (up to 4 characters, lef justified)

Byte 6 — Sortable, Non-sortable Hollerith Indicator for Dispatcher call itself only.

1 = Sortable 0 = Non-sortable

(For J-2, Hollerith is sortable; SCAMP uses non-sortable Hollerith)

Byte 7 — Number of entries in this call excluding this control word (integer).

Example:

Byte	0	1	2	3	4	5	6	7
	P	I	P	1			1	5

The program name is PIP1, the call is in Hollerith, and this call will have 5 parameter words following the control word.

(Blank bytes should be Hollerith blanks, not zeros.)

Only one I/O function may be requested in a call at a time. The parameters for this call are specified by the desired function. The value type number (byte 6) specified by the I/O function forma must always be included. (Sec. 1.2.2). If an additional number i byte 7 is required, it will be specified by the value type number byte 6. If this means that a succeeding word contains related information, the current word will so indicate by means of the val type.

Example:

F	I	L	E			6	6
M	E	S	A	G	E		

The I/O function is FILE (declaring an input and/or output file for a particular type of I/O equipment), the name of the file is located in the next word (2-8 characters, left justified), and the file name is six characters long. These two words must be consecutive. File names must be 6 characters, including trailing spaces.

Examples of Complete I/O Calls

A FILE call for teletype:

Byte IØCALL Entry	0	1	2	3	4	5	6	7	
0	P	I	P	1			1	8	
1	F	I	L	E			6	6	*
2	M	E	S	A	G	E			*
3	U	N	I	T			0	8	
4	F	Ø	R	M			5	C	
5	I	N	L	Ø	C		1		*
6 (1)						14	44	48	*
7	N	U	M	W	D	S	1		
8 (2)							20_8		*

Entry 4, FØRM, indicates that the data to be transferred will be in sortable (core 7090) Hollerith form. The call itself does not have to be in the same type of Hollerith as the data it refers to in the file. Entries 6 and 8 will be non-Hollerith entries. Entry 6 (1) can be set (in J-2) by using the LØC procedure to get the address of the table for use by the I/O file. (Example: IØCALL(6) = LØC (TABLE1)$). In JTS, it will not be necessary to use LØC, since the table address can be gotten directly.

(Example: IØCALL(6) = TABLE1 $)
Entry 8 (2) will require an integer setting.
(Example: IØCALL (8) = 16$ (20_8).

Entry 5, the INLØC parameter, is optional for any of the FILE calls. If it is not specified the INLØC will be set to zero. Whether or not

*Entries must be paired together in order.

INLØC is used, however, you may employ the CØREIX parameter in the MOVE call to specify the absolute location of the data.

It should be noted that several FILE calls (names) can be used for one data area in a program. Also several data areas can use one FILE call by means of the CØREIX (MOVE) procedure described above. That is, location references can be changed each time the MOVE is given.

The NUMWDS entry (7, 8) is optional, since the number of words to b transferred can be specified in the specific transfer reqest (MOVE)

A call to transfer teletype input into the object program:

Byte	0	1	2	3	4	5	6	7	
Entry 0	P	I	P	1			1	5	
1	M	Ø	V	E			6	6	*
2	M	E	S	A	G	E			
3	I	N	P	U	T		4		
4	N	U	M	W	D	S	1		*
5								20_8	

The NUMWDS parameter must be declared either in the FILE call or in the MOVE. However, the NUMWDS parameter is not functional for tele type input, since the number of words actually input from TTY (up t 20) will be transferred to the program. Therefore, it is important to make the table area for such inputs at least the size of a maximum TTY input (i.e., currently 20).

A call to transfer teletype output from the object program:

Byte	0	1	2	3	4	5	6	7	
Entry 0	P	I	P	1			1	5	
1	M	Ø	V	E			6	6	*
2	M	E	S	A	G	E			
3	Ø	U	T	P	U	T	4		
4	N	U	M	W	D	S	1		*
5								10_8	

The same file name refering to one program table, has been used for both input and cutput of teletype messages. This practice is not recommended, however, and separate tables should be used for input and output. This is because it will be possible to generate outputs while in the process of scanning input, and the two functions should not interfere.

The NUMWDS parameter (entries 4, 5) is optional; it is not needed if specified in the FILE call. (The MOVE call, NUMWDS, will, however, override the FILE entry of same.)

A FILE call for drum space:*

Byte	0	1	2	3	4	5	6	7	
Entry 0	P	I	P	1			1	8	
1	F	I	L	E			6	6	*
2	D	R	M	W	R	T			
3	I	N	L	ø	C		1		*
4					40	23	33	8	
5	U	N	I	T			0	4	
6	D	R	M	L	ø	C	0	0	
7	N	U	M	W	D	S	1		*
8								62_8	

Entry 3, the INLØC parameter, is optional for any of the FILE calls. If it is not specified, the INLØC will be set to zero. Whether or not INLØC is used, however, you may employ the CØREIX parameter in the MOVE call to specify the absolute location of the data.

*For the initial TSS system, an object program may ask for the use of up to 8k of drum space for its own operational use. This drum space will only be available on the Dator drum (unit 04). Entry 4 is set by using LOC (Table) in J-2.

The I/O calls to transfer to and from drum is similar to those shown for teletype, except that an additional parameter may be added to modify the drum or core location (INLØC). (The core location (i.e., TTY input or output table) may also be modified for teletype I/O

*Drum space for object programs is currently unavailable; I/O calls for drum by an object program will be recognized.

transfer calls.)

C	ø	R	E	I	X	1	
							50_8

*Number t
added to
INLØC ad
(core) fo
transfer

D	R	U	M	I	X	1	
							50_8

*Number t
added to
DRMLØC a
(drum) f
transfer

A FILE call for tape:

Byte	0	1	2	3	4	5	6	7
Entry 0	P	I	P	1			1	8
1	F	I	L	E			6	6
2	ø	1						
3	I	N	L	ø	C		1	
4								40233_8
5	U	N	I	T			0	3
6	F	O	R	M			5	C
7	N	U •	M	W	D	S	1	
8								120_8

*FILE nam
correspo
LOAD con
file ref
*

*

The calls for transfers are the same as shown for teletypes.

The INLØC parameter is optional for any of the FILE calls. If it is not specified, the INLØC will be set to zero. Whether or not INLØC is used, however, you may employ the CØREIX parameter in the MOVE call to specify the absolute location of the data.

It should be noted that only one record at a time can be transferred using a MOVE call for tape. For each record desired to be transferr

in or out, a MOVE call must be given. The number of words in the record are specified by the NUMWDS parameter.

The INSTAT is to be used with the MOVE or TAPMUV declaration. WDSIN is used only with the MOVE declaration. Neither is used with the FILE declaration.

The FILE call for tape should include an entry of FORM (Entry 6). This specifies the word format on tape to be in binary or Hollerith. If FORM is not included in the FILE call, the Dispatcher will assume that binary is desired.

Additional Parameters for Tape FILE Call

Two additional parameters may be used in a tape FILE call by an object program. These will permit an object program to ask for a specific tape reel number to be mounted on a tape drive, and have the tape file protected.

Tape Reel Number

Byte	0	1	2	3	4	5	6	7
Entry A	R	E	E	L			6	4
B	1	2	3	4				

*

If the REEL parameters are omitted (both entries A and B), a blank tape will be assumed for mounting. Also, if the reel number entry (B) is all zeros (this whole word must be zeros not blanks), the operator will mount a blank tape. The user will then receive a printout on his teletype as follows:

$FILE AAAA DRIVE XX REEL XXXX (The reel number, if a blank tape, will be inserted by the operator.)

This provides the user with information about what tape reel has been given to him and its drive location. In the event that the system halts during object program operation and is restarted, the user should determine whether or not the "mount" tape (used by the object program) should be rewound or not. This will depend on how his program operates and the validity of partial tape output. If the tape is to be rewound whenever the program starts, a TAPMUV call should be given by the object program. If rewinding is optional, the object program should query the user whether to rewind, write end-of-file, etc. It will also be possible for the user to notify the

*Entries A and B must appear together in sequence.

operator at the time of program restart to rewind the tape, if necessary.

File Protect

If the tape reel to be mounted should be file protected, the following optional entry should be included in the tape FILE call:

Byte	0	1	2	3	4	5	6	7
	P	R	∅	T	E	K	0	1

If this entry is omitted, or byte 7 is a zero (instead of a one), or the FILE call specifies a blank tape, then the tape will not be file protected.

A call to write end-of-file on tape:

Byte	0	1	2	3	4	5	6	7	
Entry 0	P	I	P	1			1	3	
1	T	A	P	M	U	V	6	6	*
2	∅	1							*
3	∅	P	T	I	∅	N	0	4	

This call can be used with various options to backspace, rewind, set density, write end-of-tape, etc.

Drum and tape files should be "defiled" if no longer necessary by the program.

Example:

Byte	0	1	2	3	4	5	6	7
	P	I	P	1			1	2
	D	E	F	I	L	E	6	6
	∅	1						

Note: An object program can be written to use tapes (and drums) optionally. That is, the FILE and MOVE calls do not have to be operated in the program code. However, if tapes will be used by the program for a particular run, it will be necessary to specify tape requirements in the LOAD (Executive) command or else the object program will be terminated by the Dispatcher.

B. I/O Call Tables

It is convenient to establish one or more tables to contain the I/O call information. To avoid confusion and minimize housekeeping, certain types of frequently used calls can have separate tables. For example, the MOVE calls, shown in the previous examples, would only require two entry changes for changing from input to output and vice versa. However, changing from FILE to MOVE is a complete change. It appears, then, that I/O calls that are used often and which vary radically in format can be conveniently handled by separate tables.

Normally, FILE will not be called for except initially in the program; however, MOVE will be used very often for teletypes as well as tape and drum transfers. The teletype MOVES may well be separated from the other I/O MOVES.

Since the Dispatcher calls involve both Hollerith and integer parameters, the I/O call table should be declared as having two kinds of items as follows: (J-2)

```
TABLE CALL R 10 1 $
  BEGIN
    ITEM CALLH 8 0 0 N S
    ITEM CALLI 48 S 00 N $
END
```

C. Calling the Dispatcher

When an object program has set up its I/O call parameter table, it can then go to the Dispatcher for service. To do this, it is necessary to go into direct code (for J-2) as follows:

```
    DIRECT $
ASSIGN A = PLACE $
BUC              312'
```

Cols. 8-10 JOVIAL Cols. 24-27 - An item may be equated to this absolute address.

PLACE is an arbitrary integer item which has been set previously to the address of the I/O call parameter table. (PLACE = LØC (CALL)$). Operation of the object program will be returned to the next instruction following the BUC to the Dispatcher address when the I/O request has been satisfied.

D. I/O Call Errors

Any mistakes in the I/O call format, or if the I/O call location give to the Dispatcher is in error, will be reported on the user's teletyp (and on a dump tape), and the object program will be terminated.

E. Tape Maintenance

Additions to the Dispatcher calls are being planned whereby, the object program can specify appropriate action in cases of tape parit end-of-file, end-of-tape, etc. This will involve tape demounting, labeling, and remounting functions by the computer operators.

6. Multiple Teletype Channel Assignment For One Object Program

Object programs, which are written to service several teletype stations simultaneously, can inform the system via Dispatcher calls which teletyp stations are to be joined to the initial program channel (program originator). The following Dispatcher call is to be used for "joining" a teletype station to the originator's program:

Byte	0	1	2	3	4	5	6	7
Entry 0	P	I	P	1			0	1
1	J	Ø	I	N			0	XX_8

- TTY stat number

Entry 0 is the standard Dispatcher control word containing the program name (up to four characters), in byte 6 is an indication of the Hollerit code used in the call (1 = Sortable, 0 = Non-sortable, and byte 7 indica one word in the remainder of the call).

Entry 1 contains the parameter JOIN, a zero in byte 6 (indicating that byte 7 contains the value for JOIN) and an octal number specifying the desired teletype station to be joined. The JOIN call should be repeated for each station desired, and the originating teletype station number should also be joined, in order for the object program to know its channel address.

Teletype stations which are joined, can all communicate with the originating object program. However, the program must be properly prepared to handle input and output for several stations. Joined channels will be initially operating in a GO status; however, normal system commands, e.g., STOP, QUIT, etc. will be honored for that individual station. Once a joined station has QUIT independently, it cannot be rejoined except by repeating the JOIN procedure again from the originating station. DBUG operations can only be performed by the originating station, not the joined stations.

Object programs which will handle joined stations should be prepared to query the originating station (via teletype) as to which station numbers (including its own) should be joined. This is an initializing procedure which can be dealt with conveniently using the teletype. Provision may also be made to re-join a station which has QUIT, by creating a sepcial command input to the object program.

Note: This call is not for use with object programs which are designed for only one TTY station (user).

7. Rescue

A Dispatcher call will be available for a special non-I/O use, which will permit an object program system to take responsibility for certain types of FIX errors. That is, under certain FIX error interrupt conditions, the information will be relayed to the object program and control branched to a designated error entry location in the object program.

Byte	0	1	2	3	4	5	6	7
Entry 0	P	I	P	1			0 1	2
1	R	E	S	C	U	E	1	
2								60010_8

Entry 0 is the standard control word for Dispatcher calls containing a program name (up to four characters), byte six indicates whether the Hollerith characters in the call itself are sortable (1) or non-sortable (0), byte seven indicates that two more words are in the call. Entry 1 must be set as indicated.

Entry 2 will contain the absolute location of an error routine in the object program, where the system will branch to after a FIX error interrupt.

When the Dispatcher receives the RESCUE call at the start of object program operation, it will set a status item indicating that this prog will take responsibility for handling FIX errors caused by the object program. At the time an error occurs, the system will do the followin

a. Set the object program panel environment storage to the values existing at time of interrupt. (These include the Accumulator, B Reg, Live Reg, Index Regs, etc.)

b. The same interrupt (error) environment will be reestablished in th machine registers.

c. The system will branch to the error routine location in the object program specified in the RESCUE call.

d. The system will insert a value, indicating the type of error, into the register following the error entry location specified in the RESCUE call. The error value will appear in byte 7 and have the following meanings:

1 = Illegal branch into FIX. (This error will not permit saving t interrupt panel environment, and the panel will reflect the environment of the object program at the time it last went to the system.)

2 = Illegal instruction.

3 = Illegal address reference.

4 = Illegal division.

5 = Program halt.

In all cases, the error address may be off by one register (+1).

e. In addition to the above, the system will print out an error messa on the user's teletype.

Note: Object programs which use RESCUE must be prepared to handle these error conditions in one way or another. For example, in IPL it may go into a trace dump, or if the error is too serious to continue operation, the object program should notify the user to quit.

8. Teletype Control Characters

Teletype code will be converted by the system to and from Hollerith for object programs. In addition to the normal printable characters in the code set, certain control characters for teletype will also be available. These are not standard Hollerith characters and are set to specific octal values.

End-of-message 77_8

Line Feed & Carriage Return 32_8

Input messages should terminate with a byte containing 77_8; this will cause the TTY output routine to stop character transfer after that point. Three limits on TTY output exist, as far as the number of characters actually transferred is concerned. The first type of cut-off point is the end-of-message character 77_8, as described above. The second type of cut-off is the number of words specified in the I/O call (NUMWDS). Finally, the maximum limit of the output drum buffer is twenty words or ninety-six characters. This limit includes control characters which must be added in the teletype conversion and no more than eighty characters should normally be sent at one time. If the character limit is exceeded, the excess will be lost (not transferred). This output limit of 96 characters will be changed when the peripheral computer is installed (PDP-1).

The system teletype output routine currently prefixes all output messages with a line feed and carriage return. This means that object programs do not have to provide this control character intheir messages, except for formatting within an individual output message. The system will automatically line space individual messages and the object program may line space within individual messages using the line feed/carriage return character (32_8).

If messages from the object program are generally long, print them out in short bursts to avoid losing characters. When in doubt, print it out!

9. Program Checkout

Object programs written in J-2 can be partially checked-out by using the Q-32 JOVIAL Support System. This will permit satisfactory parameter testing to be accomplished. A convenient method of testing the object program is to replace all exits to the Time-Sharing System (Executive or Dispatcher) by the procedure call for the JOVIAL system

symbolic table and item dump and use simulated inputs. This will re the cirtical input and output functions for debugging purposes. Aft this type of testing is complete, the program may then be cycled by Time-Sharing System for final checkout.

10. JTS (JOVIAL for Time-Sharing System)

There will be several changes in the Executive command (teletype ope for calling the JTS compiler. These changes will be described in th documentation of JTS when it becomes available for use. This should affect object program preparation.

11. Object Program Core Location

The initial phase of the Time-Sharing system will not provide space-sharing of core memory by object programs until memory protection is available. Object programs will be given absolute core address assig ments by the programmer prior to (for SCAMP) or at binary tape loadi time (J-2 and other relative-addressing compilers). The legal limit for ORGing an object program run from 40000_8 to 172000_8. The core s declared by a program (within the above limits) must include all tab areas.

APPENDIX A

```
LCC                5-1-63                    8  DISPATCHER CALLS
''
1          FORMAT OF CALLS TO THE DISPATCHER                          2  0

1.1   CALLING SEQUENCE CONVENTIONS                                   3  3

      WHEN A PROGRAM MAKES A CALL TO THE DISPATCHER IT WILL LEAVE    4  2
      IN THE ACCUMULATOR THE LOCATION OF ITS PARAMETER REQUEST       5
      TABLE.                                                         6
         THE DISPATCHER WILL SAVE APPROPRIATE REGISTERS AND          7  1
      RESTORE THEM FOR SYSTEM PROGRAMS,STORE THEM FOR LATER          8
      RESTORATION FOR NON-SYSTEM PROGRAMS.                           9



1.2   CALLING PARAMETER TABLE FORMAT                                10  3

1.2.1 THE FIRST WORD IN THE TABLE IS OF THE FOLLOWING FORMAT.       11  2

      BYTE  0-3  NAME OF CALLING PROGRAM - 1 TO 4 CHARACTERS,       12  1
                                           LEFT JUSTIFIED           13
      BYTE    6  TYPE OF  HOLLERITH  1=SORTABLE   0=NON SORTABLE   1311
      BYTE    7  NUMBER OF WORDS IN THIS TABLE, NOT                 14  1
                 INCLUDING THIS WORD                                15

1.2.2 THE INFORMATION IN THE MAIN BODY OF THE TABLE IS OF THE       16  2
      FOLLOWING FORMAT.                                             17

      BITS  0-35 PARAMETER NAME- 1 TO 6 CHARACTERS,LEFT JUSTIFIED   19  1

      BITS 36-41 VALUE TYPE- INTEGER WITH THE FOLLOWING VALUES      20  1
           0=VALUE IS INTEGER 42-47 OF THIS WORD                    21
           1=VALUE IS INTEGER IN NEXT WORD, RIGHT JUSTIFIED         22
           3=VALUE IS FLOATING POINT NUMBER IN NEXT WORD            23
           4=THERE IS NO VALUE WITH THIS PARAMETER NAME             24
           5=VALUE IS 1 HOLLERITH CHARACTER BYTE 7 THIS WORD        25
           6=VALUE IS 2-8 HOLLERITH CHARACTERS LEFT JUSTIFIED       26
             IN NEXT WORD. NUMBER OF CHARACTERS IS IN BYTE 7        27
             THIS WORD (A BINARY INTEGER).                          28
           7=VALUE IS AN ARBITRARY ARRANGEMENT OF INFORMATION IN    29
             NEXT N WORDS. N IS A BINARY INTEGER IN BYTE 7 OF THIS  30
             WORD                                                   31


1.3   IN ANY REQUEST THERE CAN BE ONLY ONE OF THE PARAMETER NAMES   32  2
      PRECEEDED BY A * BELOW.                                       33
      THE ORDER OF THE PARAMETER NAMES IN THE CALL IS NOT           34  1
      SIGNIFICANT.                                                  35
```

THE PARAMETER NAMES ARE ALL OPERATED ON AS 6 CHARACTER ITEMS, LEFT JUSTIFIED BLANK CHARACTERS, NOT ZEROES MUST FILL IN UNUSED CHARACTERS

WHEN A PROGRAM IS REENTERED IT WILL BE AT THE LOCATION IN THE PROGRAM REGISTER UPON ENTERING THE DISPATCHER

2.0 THE FOLLOWING LIST DEFINES THE VARIOUS REQUSTS TO THE DISPATCHER AND THEIR FUNCTION

2.1 FILE - THIS IS USED TO RESERVE I/O EQUIPMENT FOR PROGRAMS AND TO MINIMIZE TRANSMISSION OF INFORMATION ON MOVE CALLS. IT ALSO MAY RESERVE A BLOCK OF SPACE ON DRUMS. ALL FILE DECLARATIONS SHOULD BE MADE AT THE BEGINNING OF A PROGRAMS OPERATION.

2.2 DEFILE - CANCELS ALL RESERVATIONS AND INFORMATION STORAGE CAUSED BY FILE

2.3 TAPMUV - USED TO MANIPULATE TAPE, BUT NOT FOR READING TAPE.

2.5 ASSIGN - ASSOCIATES FILE NAME WITH LOGICAL I/O UNIT

2.6 MOVE - CAUSES INFORMATION TO BE TRANSFERRED TO OR FROM CORE ACCORDING TO INSTRUCTIONS IN FILE DECLARATION

2.7 SCROLL - CAUSES INFORMATION WHICH HAS BEEN PUT ON DRUMS BY OBJECT PROGRAM TO BE ENTERED INTO INVENTORY

2.8 TOCORE - CAUSES ENTTIES IN INVENTORY TO BE BROUGHT TO CORE

2.9 TODRUM - CAUSES INVENTORIED ENTITY IN CORE TO BE PLACED ON DRUM

3.0 DELETE - CAUSES PROGRAM OR TABLE TO BE REMOVED FROM INVENTORY

3.1 LOAD - CAUSES PROGRAM ON INDICATED BINARY TAPE TO BE RELOCATED AND PUT ON DRUM

3.2 DIRECT - CALL CONTAINS 3 WORD CALL TO I/O PACKAGE IN I/O PACKAGE FORMAT

3.3 CON - CALL FOR USE OF SCHEDULER TO DETECT CORE CONFLICTS

3.4 IOCINT - ENTRY FOR IO COMPLETE INTERRUPT

3.5 JOIN - PERMITS ADDITIONAL TTY STATION TO COMMUNICATE WITH AN OBJECT PROGRAM .

3.6 RESCUE - ENABLES AN OBJECT PROGRAM TO RETAIN CONTROL AFTER A FIX INTERFACE ERROR INTERRUPT .

	PARAMETER NAME	COMMENT NUMBER	VALUE TYPE	SIZE	VALUE
4.1	* FILE		6	6	NAME OF FILE
	UNIT		0		0=CARDREADER 1=PRINTER 2=PUNCH 3=TAPE 4=DRUM 5=INPUT MEMORY 6=I/O REGISTER 7=CONSOLE TYPWRITER 8=TELETYPE 9=REMOTE HIGH SPEED DEVICE
	FORM	(1)	5		B=BINARY H=HOLLERITH (NON-SORTABLE) C=7090 CORE HOLLERITH (SORTABLE)
	INLOC		A) 1		LOCATION OF FIRST CORE REGISTER TO TRANSFER-BINARY
			B) 6	6	NAME OF PROGRAM OR TABLE IN SYSTEM
	DRMLOC	(2)	A) 7	1	BITS 18-23 0=DRUM A 1=DRUM B 4=DATOR DRUM BITS 30-34 = DRUM FIELD BITS 35-47 = DRUM ADDRESS
			B) 6	6	NAME IN INVENTORY
			C) 0		0=RESERVE DRUM REGISTERS IN FILE NAME EQUAL TO NUMWDS
			D) 1		RESERVE DRUM REGISTERS IN FILE NAME EQUAL TO INTEGER IN NEXT WORD

	DENSTY	(3)	5		H=HIGH L=LOW
	NUMWDS	(4)	1		NUMBER OF WORDS TO BE MOVED
	TAPE	(5)	0		LOGICAL TAPE NUMBER
	REEL	(6)	6	4	TAPE REEL NUMBER (OR DECK)
	PROTEK	(7)	0		0= FILE PROTECT TAPE 1 = DO NOT FILE PROTECT

COMMENTS

(1) NOT USED IF UNIT EQUALS DRUM, INPUT MEMORY
(2) USED ONLY WITH DRUM
(3) USED ONLY WITH TAPE
(4) NOT NEEDED IF DRMLOC VALUE TYPE EQUALS 6 OR 0 OR IF NUMWDS PROVIDED IN MOVE REQUEST
(5) OPTIONAL FOR USE WITH TAPE
(6) OPTIONAL FOR TAPE (OR CARD READER)
(7) OPTIONAL FOR USE WITH TAPES
IF A TAPE FILE IS REQUESTED AND NO REEL IS SPECIFIED A NON FILE PROTECTED BLANK WILL BE MOUNTED

PARAMETER NAME	COMMENT NUMBER	VALUE TYPE	SIZE	VALUE
4.2 * DEFILE		6	6	NAME OF FILE TO BE EXPUNGED
4.3 * TAPMUV		6	6	FILE NAME IDENTIFYING LOGICAL TAPE

	OPTION	0		4=WRITE END OF FILE	237	3
				5=WRITE END OF TAPE	238	
				6=REWIND	239	
				7=BACKSPACE A RECORD	240	
				8=BACKSPACE A FILE	241	
				9=SET HIGH DENSITY	242	
				10=SET LOW DENSITY	243	
				DENSITY CAN BE CHANGED ONLY	2432	1
				ON TAPES IN REWIND POSITION	2434	
	TNSTAT	0		SEE SAME PARAMETER UNDER	2436	3
				MOVE FOR VALUES	2438	
4.5	* ASSIGN	6	6	FILE NAME	244	5
	TAPE	A. 0		LOGICAL TAPE NUMBER	245	3
		B) 4		IF VALUE TYPE EQ 4 DISPATCH-	2452	1
				ER ASSIGNS TAPE AND LEAVES	2454	
				LOGICAL TAPE NUM IN VALUE	2456	
	PNAME	6	6	PROGRAM NAME OF PROG USING	2458	3
				FILE IF NOT SAME AS CALLING	24585	
				PROGRAM	24586	
	TTYCHN	0		TELETYPE NUM FIRST=0	246	3
4.6	* MOVE	6	6	FILE NAME	247	5
		0		RELATIVE LOCATION IN SYSTEM	248	1
				TABLES OF ENTITY TO BE	249	
				MOVED	250	

237 3	IDENT	(1)	6	4	IDENT CHARACTERS FOR TAPE TRANSFERS - CAUSES THIS READ OR WRITE TO BE IN IDENT MODE
238				
239				
240				
241				
242				
243	NUMWDS	(3)	1		NUMBER OF WORDS TO BE MOVE
Y 2432 1				
N 2434				
	INPUT	(2)	4		CAUSES TRANSFER TO CORE
2436 3				
2438				
	OUTPUT	(2)	4		CAUSES TRANSFER FROM CORE
	COREIX	(1)	1		NUMBER TO BE ADDED TO INL OF FILE DECLARATION TO ESTABLISH MODIFED INLOC F. THIS TRANSFER ONLY
244 5				
245 3	DRUMIX	(1)	1		NUMBER TO BE ADDED TO DRUI REGISTER AS DEFINED OR REQUESTED IN DRMLOC OF FI. DECLARATION
H-2452 1				
2454				
2456				
2458 3	TTYCHN	(4)	0		TELETYPE NUM FIRST=0
24585				
24586				
	TNSTAT		0		THIS ITEM SET BY DISPATCHEF AT COMPLETION OF MOVE IF PROVIDED BY CALLING PROGRAI 0=INITIAL DISP SETTING 1=REQUEST IN STACK 2=REQUEST IN OPERATION 3=REQUEST COMPLETE 4=READ REQUEST FOUND EOF, OPERATION COMPLETE 5=READ REQUEST FOUND EOT, OPERATION COMPLETE 6=IF TAPE WRITE WROTE OVER TAPE END
246 3				
247 5				
M248 1				
249				
250				

				IF DRUM READ OR WRITE	26444
				STEPPED TO ILLEGAL FIELD	26446
				7=REQUEST RESULTED IN	26450
				ILLEGAL TAPE MOTION	26452
				8=AN UNFIXABLE PARITY	26454
				OCURRED	26456
				9 REQUEST REJECTED-ILLEGAL	26455
				(THESE VALUES NOT SET)	26458
				(ON TELETYPE MOVES)	26460
	WDSIN	1		DISPATCHER INSERTS NUMBER	2647 3
				WORDS READ FROM TAPE IF	2648
				PROVIDED	2649
			COMMENTS		265 3
	(1) OPTIONAL				266
	(2) EITHER INPUT OR OUTPUT -BUT NOT BOTH- ARE USED				267
	IF USED OVERRIDES FILE				26701
	(3) NOT NEEDED IF GIVEN IN FILE DECLARATION,				26705
	(4) OPTIONAL - IF USED WITH FILE DECLARED AS TTY IT WILL				26707
	BY TTY CHANNEL USED (THIS MOVE ONLY)				26709
4.7	* SCROLL	6	6	FILE NAME IDENTIFIS AREA	2671 5
				NOW ON DRUMS TO BE GIVEN	2673
				INDEPENDENT IDENTITY	2675
	NAME	6	6	NEW NAME OF ENTITY	2677
4.8	* TOCORE	1		PQU CHANNEL OF PROGRAM	269 5
4.9	* TODRUM	1		PQU CHANNEL OF PROGRAM	270 5

26444				
26446	5.0 * DELETE	6	6	NAME IN INVENTORY
26450				
26452				
26454				
26456				
26455				
26458	5.1 * LOAD	1		CHANNEL IN PQU OF PROGRAM TO BE LOADED
26460				
2647 3				
2648				
2649	5.2 * DIRECT	7	3	NEXT THREE WORDS CONTAIN I/O CALL IN FORMAT OF I/O PACKAGE. THERE WILL BE LEGALITY CHECKS ON THIS REQUEST. USE ONLY AFTER CHECKING WITH SYSTEM DESIGNERS.
265 3				
266				
267				
26701				
26705				
26707				
26709	5.3 * CON	1		INTEGER IS CHANNEL NUMBER IN QUEUE OF PROGRAM TO BE CHECKED FOR CONFLICTS. DISPATCHER INSERTS IN BYTE OF WORD CONTAINING CON 0=NO CONFLICT 1=PGM(S) IN CONFLICT BUT CAN NOW BE MOVED 2=CONFLICT PROGRAM(S) NOW DOING I/O 3=CONFLICT WITH PROGRAM IN PCUR
2671 5				
2673				
2675				
2677				
269 5	5.4 * IOCINT	4		THIS INDICATES AN IO COMPLETE INTERRUPT - USED ONLY BY SYSTEM
70 5				
	5.5 * JOIN (1)	0		TELETYPE NUM, FIRST=0

COMMENTS

(1) TELETYPE STATION (CHANNEL) NO. IS ONE HIGHER THAN TELETYPE NUMBER REFERENCES IN DISPATCHER CALLS E.G., STATION 1 = 0.
JOIN CALL MUST BE GIVEN SEPARATELY FOR EACH ADDED STATION, INCLUDING ORIGINAL STATION .

5.6 * RESCUE (1) 1 ERROR RECOVERY ADDRESS IN OBJECT PROGRAM LOCATION.

COMMENTS

(1) SYSTEM WILL BRANCH TO SPECIFIED RECOVERY ADDRESS AFTER ERROR INTERRUPT AND DEPOSIT ERROR INDICATOR IN BYTE 7 OF NEXT REGISTER .
VALUE OF BYTE 7 = ERROR CONDITION-
1 = ILLEGAL BRANCH TO FIX
2 = ILLEGAL INSTRUCTION
3 = ILLEGAL ADDRESS REFRNCE
4 = ILLEGAL DIVISION
5 = PROGRAM HALT

FIN

APPENDIX B

TELETYPE (MODEL 28) AND HOLLERITH CHARACTER SETS FOR TSS-1

28 TTY CHARACTER	HOLLERITH CHARACTER		SORTABLE (IN STORAGE)		NON-SORTABLE (ON TAPE)
Space	Blank	60	110 000	20	010 000
ø	0	00	000 000	12	001 010
1	1	01	000 001	01	000 001
2	2	02	000 010	02	000 010
3	3	03	000 011	03	000 011
4	4	04	000 100	03	000 100
5	5	05	000 101	05	000 101
6	6	06	000 110	06	000 110
7	7	07	000 111	07	000 111
8	8	10	001 000	10	001 000
9	9	11	001 001	11	001 001
:[1]	=	13	001 011	13	001 011
'	'	14	001 100	14	001 100
&[1]	+	20	010 000	60	110 000
.	.	33	011 011	73	111 011
)	)	34	011 100	74	111 100
—	—	40	100 000	40	100 000
$	$	53	101 011	53	101 011
#[1]	*	54	101 100	54	101 100

[1] These teletype characters will be replaced by the Hollerith equivalents.

28 TTY CHARACTER	HOLLERITH CHARACTER		SORTABLE (IN STORAGE)		NON-SORTABLE (ON TAPE)
/	/	61	110 001	21	010 001
,	,	73	111 011	33	011 011
(	(	74	111 100	34	011 100
A	A	21	010 001	61	110 001
B	B	22	010 010	62	110 010
C	C	23	010 011	63	110 011
D	D	24	010 100	64	110 100
E	E	25	010 101	65	110 101
F	F	26	010 110	66	110 110
G	G	27	010 111	67	110 111
H	H	30	011 000	70	111 000
I	I	31	011 001	71	111 001
J	J	41	100 001	41	100 001
K	K	42	100 010	42	100 011
L	L	43	100 011	43	100 100
M	M	44	100 100	44	100 101
N	N	45	100 101	45	100 110
O	O	46	100 110	46	100 111
P	P	47	100 111	47	101 000
Q	Q	50	101 000	50	101 001
R	R	51	101 001	51	101 010
S	S	62	110 010	22	010 011
T	T	63	110 011	23	010 011
U	U	64	110 100	24	010 100
V	V	65	110 101	25	010 101
W	W	66	110 110	26	010 110
X	X	67	110 111	27	010 111
Y	Y	70	111 000	30	011 000
Z	Z	71	111 001	31	011 001

28 TTY CHARACTER	SORTABLE (IN STORAGE)		NON-SORTABLE (ON TAPE)		
!	55	101 101	55	101 101	Model 28 Teletype characters only.
?	36	011 110	36	011 110	
BELL[2]	77	111 111	77	111 111	
LF/CR	32	011 010	32	011 010	
;	76	111 110	76	111 110	
"	56	101 110	56	101 110	

[2] End-of-message.

APPENDIX C

TIME-SHARING SYSTEM OUTPUT MESSAGE LIST

The following list describes the current output messages that the Time-Sharing System (TSS-1) will produce on the user's teletype under various conditions. In some cases, the exact wording of a message may be changed from the form shown below, but the content will remain essentially the same.

A. Executive Program Source - Teletype Input Interpreter

1. TIME SHARING SYSTEM ON THE AIR - Printed out on all teletypes when the system starts operating.

2. $TRNSMSN ERR, REPEAT - Teletype input message transmission incorrect; repeat input.

3. $MSG IN - Any Executive command which has been received and processed will produce this acknowledgement.

4. $WAIT - When a delay is anticipated in processing a user's request (Executive command), due to operator activity, the system will notify the user to "WAIT." This will primarily occur during tape mounting actions.

5. $CANCLD - In response to CANCEL command (3 line feeds), current input message has been discarded.

6. $COMND ILLEGAL FOR PGM STATUS - The Executive command received is not legal for the current status of the user's program, e.g., a program STOP cannot be given if the program has not been given a GO after loading.

7. $BLK FULL, PROC AS EOM - If a long input message fills the input buffer's capacity, the system will process the input characters received up to that point without the need for an EOM (BELL). (If a long input is desired, enter the later characters slowly, so that

the system can process the first buffer-full portion before succeeding characters clobber prior input.)

8. $ILLEGALITY IN PARAMETER NR___, SENT AS___. - The value of an Executive comman parameter is illegal. The comma must be repeated with legal parameter values.

9. $INSFCNT NR PRMETERS - The Executive command received does not contain the proper numb of parameters required.

10. $NO RECVR FOR DIAL - The input parameter for the DIAL command does not specify a legal channel number address, e.g., if the sender's channel number is given as the address.

11. XX TO XX (MESSAGE) - This is the header which precede a DIAL message output to a recei The header indicates both the sender's channel number and the addressees channel number.

12. $TRY AGAIN - If user asks for DBUG function before his object program is available (i.e., loaded) repeat request after $LOAD OK is recei

NOTE: The system will refer all Executive mode inputs which it does not understand to the Debugging program. If DBUG cannot accept the input, it will print out an error message (e.g., ERR1, ERR 7, etc.). See TM-1126/004/00, page 13, for DBUG error messages.

B. Executive Program Source - FIX Interface

All object programs which are trapped through the FIX interface because of an illegality will be set to a status of "logical conclusion," with one of the error printouts listed below. At this point, the program may be restarted by setting the Live Register (via DBUG) to an appropriate location in the object program, and giving the GO command.

1. PGM TERMINATED ON ILLEGAL ADDRESS REFERENCE AT XXXXXX - System has trapped an object program which exceeds the upper or lower limits of the computer ($30 > x > 65k$). The printout gives the Program Counter (address) at time of interrupt, which may be off by one register.

2. PGM HUNG ON AN ILLEGAL DIVISION AT XXXXXX - The object program has been interrupted when trying to perform an illegal division. Address indicates location of division instruction.

3. PGM HALTED AT XXXXXX - An object program which executes a halt (any register with all zeros) will be trapped by the system. Object programs, when logically concluded, should branch to the system (BUC 303').

4. PGM TERMINATED ON AN INDETERMINATE ERROR - If an unaccountable error has caused FIX to interrupt the system, the currently operating object program user will be notified of the situation with this printout.

5. PGM WAS TRAPPED ON ILLEGAL REENTRY LOCATION AT XXXXXX - If the branch address in the Live Register for an object program exceeds the upper or lower system limits ($16k > x > 64k$), the program will be trapped. The address in the Live Reg will be indicated in the printout.

6. PGM WAS TRAPPED ON ILLEGAL INPUT DATA LOCATION - This condition is similar to 5, above, except that it applies to illegal locations specified for input data transfers (via I/O calls).

7. PGM WAS TRAPPED ON ILLEGAL DOUBLE INDEX - If double index has an illegal number of bits for its assigned value, program will be trapped.

5k).

8. PGM HUNG ON AN ILLEGAL INSTRUCTION AT XXXXXX - Any bad instruction caused by objec program error or any other reason will result in this trap and printout.

9. PGM TERMINATED ON ILLEGAL BRANCH INTO FIX - If FIX determines that it has been entered illegally, the system will terminate the currently operating object program.

10. PROGRAM CONCLUDED - This is not an error printout*, but merely notice that the object progr has completed its run and returned control to the system. It may be operated again by setting the Live Reg to the starting address, and giving the GO command.

C. Executive Program Source - Dispatcher

The Dispatcher is responsible for all I/O transfers and will generate appropriate error messages if any I/O call is in error. In addition, the Dispatcher will communicate information concerning tape handling operations.

1. $LOAD OK - Binary object program has been loaded from tape to drums in response to the LOAD command.

2. $NO LOAD - Binary object program could not be loaded from tape to drum for some reason.

3. $XX TAPES AVBL - In response to user's input request for number of tape drives available (Current maximum is five for object programs.)

4. $FILE XXXX DRIVE XX REEL XXXX - Indicates to user reel number of a "mount" tape requested by object program during its operation. This reel number should be referenced for later use of tape, if necessary

*In certain cases of program or machine error, where the system may stop operating, the operator may enter the "logical conclusion" address to continue the system. This would cause a PROGRAM CONCLUDED message to appear on the user's teletype, and would indicate that a manual termination (due to some error) was implemented. As the system evolves, this problem will be treated in a better manner.

5. BAD LOCN - Object program indicates a location (in Accumulator) for I/O call table which exceeds upper or lower system address limits (24 > x > 65k). Program will be put into STOP status.

The succeeding Dispatcher error messages will appear in a general format as follows:

AAAAAAAAAA Error Type AAAA Program Ident AAAAAAAA Location I/O Call in Error DISPATCHER CALL ERROR

The object program will be placed in a STOP status if the errors below occur. The user may be able to take corrective action (using DBUG) and the RESUME command.

6. NO FILE - No file name found to match file reference in call (e.g., a MOVE call must have a FILE declared previously). May appear as DMOVEA, currently.

7. NO NUMWD - The NUMWDS parameter is missing from both the FILE and subsequent MOVE calls. It must be in at least one of these.

8. CORE ADD - Address specified by object program for an I/O transfer exceeds upper and lower address limits of system (i.e., 24 > x > 65k).

9. ILL ID R - Illegal ID read for tape (i.e., ID given in binary).

10. NO OPTN - No OPTION parameter specified for TAPMUV call.

11. BAD OPTN - Illegal parameter value specified for OPTION.

12. IL TAPE - Object program has specified a logical tape drive for tape usage. This is legal only for system programs.

13. NO TAPES - Object program has asked for a tape drive and none are available.

14. ILL CHAN - Non-existent I/O channel specified for joining several user channels to one program.

15. BAD CHAN - Specified channel not in proper status to be joined to originator's program.

16. NO CALL - Object program has not specified location of recognizable I/O call table in Accumulator at time of branch to Dispatcher.

17. VAL TYPE - Illegal Value Type specified in call.

18. DREQ OVR - Dispatcher overloaded with I/O requests and cannot accept any more for stacking. (This situation although possible, should hopefully never occur.)

19. ILL READ - Object program has asked for an I/O read which will exceed the address limits of the object programs core space allocation.

20. DMOVEH - The INPUT or OUTPUT parameter is missing from a MOVE call.